High-Performance Big Data Management Across Cloud Data Centers

Radu Tudoran

THÈSE / ENS RENNES
sous le sceau de l'Université européenne de Bretagne

pour obtenir le titre de
DOCTEUR DE L'ÉCOLE NORMALE SUPÉRIEURE DE RENNES
Mention : Informatique
École doctorale MATISSE

présentée par
Radu Marius Tudoran
Préparée à l'unité mixte de recherche 6074
Institut de recherche en informatique
et systèmes aléatoires

High-Performance Big Data Management Across Cloud Data Centers

Thèse soutenue le 10 décembre 2014
devant le jury composé de :

Frédéric Desprez / *rapporteur*
Directeur de recherche, Inria Rhône-Alpes, France

Michael Schöttner / *rapporteur et examinateur*
Professor, Institute of Informatics, Duesseldorf University, Germany

Pierre Sens / *examinateur*
Professeur, Université Paris 6, France

Olivier Nano / *examinateur*
Principal Development Manager at Microsoft Research, ATLE, Germany

Patrick Valduriez / *examinateur*
Directeur de recherche, Inria Sophia Antipolis-Méditerranée, France

Gabriel Antoniu / *directeur de thèse*
Directeur de recherche, Inria Rennes - Bretagne Atlantique, France

Luc Bougé / *directeur de thèse*
Professeur, ENS Rennes, France

Du bist aufgehoben für einen großen Montag! Wohl gesprochen, aber der Sonntag endet nie
You are destined for a great Monday! Well said, but the Sunday never ends
Reisetagebücher, 1921, Franz Kafka

Acknowledgements

This PhD work was made possible thanks to the patience, guidance and helpful advices of my excellent supervisors Gabriel and Luc, and my close collaborator and colleague Alexandru. I am most grateful for your support and for offering me this great and enriching experience. Thank you for everything!

I would like to thank also my beloved family: Anca, Radu and Ileana, for their continuous encouragement, support and help in every step that I make. You provide me the strength that I need to go forward.

I would also like to thank the members of the jury: Olivier Nano, Patrick Valduriez and Pierre Sens and my evaluators Michael Schöttner and Frédéric Desprez for taking the time to evaluate my work and give me valuable feedback.

Kind regards go to my internship supervisors and collaborators: Götz Brasche, Olivier Nano, Ivo Santos, Hakan Soncu, Ramin Rezai Rad and Kate Keahey. I am grateful for giving me the chance to work with you and for all your mentoring advices. You have all helped me to improve my work and myself.

Many thanks go to the different contributors and collaborators. To Benoit Da Mota, Bertrand Thirion, Götz Brasche, Hakan Soncu and Pierre-Louis Xech for the great collaboration that we had in the A-Brain project. To Patrick Valduriez, Esther Pacitti, Ji Liu, Luis Pineda Morales and Olivier Nano for the short time that I had the chance to work with you in the Z-CloudFlow project. I would also like to thank Dennis Gannon, Tony Hey and Kenji Takeda for their valuable support and help in these projects and for all our interactions. I would also like to thank Kate Keahey, Pierre Riteau and Sergey Panitkin for our work within the framework of the Data@Exascale joint team.

Thanks go also to all the current and former members of the KerData team: Shadi, Houssem, Matthieu, Orçun, Tien Dat, Alvaro, Lokman, Pierre, Diana, Alexandra, Andreea, Stefan, Elena, Viet-Trung, Bogdan, Ciprian, Florin and Catalin. I happily recall our quality time together.

I would also like to thank a number of friends that were particularly supportive along this path. All my gratitude to Octavian, to whom I owe my first steps in research. Many thanks to Costin with whom I closely shared many enlightening experiences during these years. Many thanks also to my old and close friends Sebi, Marius, Nicu and Florin with whom I made together many of the steps that have taken me here.

Finally, I would like to thank all the other people that had a direct or indirect contribution to this work and were not mentioned above. Your help and support is appreciated.

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Contributions . . . 2
1.3 Publications . . . 7
1.4 Software . . . 8
1.5 Organization of the Manuscript . . . 9

Part I — **Context: The Landscape of Big Data Management on Clouds** **11**

2 Background: The Era of Big Data **13**
2.1 The Data Deluge . . . 13
2.2 Data Science: The Emergence of a New Scientific Paradigm . . . 16
2.3 Discussion . . . 18

3 Background: Cloud Computing **19**
3.1 Overview . . . 19
3.2 The Cloud Computing Paradigm . . . 21
3.3 The Cloud Storage Paradigm . . . 23
3.4 Discussion . . . 25

4 Objective: Processing Big Data on Clouds **27**
4.1 Overview of Big Data Applications . . . 27
4.2 Challenges and Issues . . . 28
4.3 Big Data Processing Models: MapReduce and Beyond . . . 31
4.4 Discussion . . . 32

5 State-of-the-Art: Big Data Management Systems for Clouds **33**
5.1 Data Storage . . . 33
5.1.1 Unstructured Data: Object-Based Storage Systems . . . 33
5.1.2 Structured Data: Distributed File Systems . . . 34
5.1.3 Structured Data: Key-Value Stores . . . 35
5.2 Data Processing . . . 36
5.2.1 MapReduce . . . 37
5.2.2 Workflows . . . 39

5.2.3 Complex Event Processing Platforms . . . 40
5.3 Transfer . . . 41
5.3.1 Static Data . . . 41
5.3.2 Real-Time Data . . . 42
5.4 Discussion . . . 43

Part II — **High-Performance Big Data Management on a Single Data Center** **45**

6 MapReduce for Bio-Informatics: The A-Brain Case Study Application **47**
6.1 Joining Genetic and Neuro-imaging Analysis . . . 47
6.1.1 Initial Motivation . . . 48
6.1.2 A-Brain: Application Description . . . 48
6.1.3 Challenges . . . 50
6.2 Towards a MapReduce Architectural Solution . . . 51
6.3 Discussion . . . 52

7 TomusBlobs: Leveraging Locality for MapReduce Applications on Azure Cloud **53**
7.1 TomusBlobs: Federating Virtual Disks for a Communication Efficient Storage 56
7.2 Leveraging Virtual Disks for Efficient MapReduce Processing . . . 58
7.3 Validation and Experimental Evaluation . . . 61
7.3.1 Cloud Storage Evaluation: TomusBlobs vs. Cloud-Provided Storage Service in Synthetic Settings . . . 61
7.3.2 Initial Experimentation with the A-Brain Application . . . 63
7.3.3 A Cost Analysis for Executing Scientific Applications on the Cloud . . . 64
7.4 Extending TomusBlobs for Efficient Workflow File Management . . . 65
7.5 Validation and Experimental Evaluation for Workflows . . . 71
7.5.1 TomusBlobs in the Context of Synthetic Workflows . . . 71
7.5.2 Using TomusBlobs to Execute a Biological Workflow Application . . . 72
7.6 Discussion . . . 73

8 Going Further: Scaling MapReduce across Multiple Data Centers **75**
8.1 Map-IterativeReduce: Handling Reduce-Intensive Workloads . . . 77
8.2 Geographically Distributed MapReduce . . . 81
8.3 Validation and Experimental Evaluation . . . 83
8.3.1 Selecting the VM Type: Impact of Multi-Tenancy on Performance . . . 83
8.3.2 Performance Gains with Map-IterativeReduce . . . 85
8.3.3 Hierarchical Multi-Site MapReduce . . . 87
8.4 Discussion . . . 89

9 Lessons Learned : Large-Scale Big Data Experiments on the Cloud **91**
9.1 A Large-Scale Experiment for Fitting Genotypes with Subcortical Brain Regions 92
9.2 Focusing on Long-Running Scientific Experiments . . . 94
9.3 Addressing Data Management Issues across Data Centers . . . 97
9.3.1 Azure-Specific Observations . . . 98
9.3.2 Beyond Azure . . . 99
9.4 Discussion . . . 102

Part III — **High-Performance Big Data Management across Data Centers** **103**

10 DataSteward: Using Dedicated Nodes for Scalable Data Management **105**
10.1 A Storage Service on Dedicated Compute Nodes . . . 107
10.2 Zoom on the Dedicated Node Selection . . . 110
10.3 Experimental Evaluation and Functionality-Perspectives . . . 113
10.3.1 Clustering Algorithm Evaluation . . . 113
10.3.2 Data Storage Evaluation . . . 115
10.3.3 Data Processing Services for a Scientific Application . . . 117
10.3.4 Going Further . . . 119
10.4 Discussion . . . 120

11 Bridging Data in the Clouds **121**
11.1 An Environment-Aware Approach for Inter-Site Transfers . . . 123
11.2 The Cost of Performance across Data Centers . . . 127
11.2.1 Cloud Data Transfer Model . . . 127
11.2.2 Efficiency in the Context of Data Management . . . 129
11.2.3 Multiple Data Center Paths Transfer Strategy . . . 130
11.3 Validation and Experimental Evaluation . . . 132
11.3.1 Evaluation of the Performance Model . . . 132
11.3.2 Data Transfer Service Evaluation . . . 134
11.3.3 The Cost–Execution Time Efficiency of Data Transfers . . . 137
11.4 Discussion . . . 140

12 Real-Time Data Management across Data Centers **141**
12.1 Evaluating Strategies for Cloud Stream Processing . . . 143
12.2 Modeling the Streaming of Data in the Context of Clouds . . . 147
12.2.1 Zoom on the Event Delivery Latency . . . 148
12.2.2 Multi-Route Streaming . . . 149
12.3 JetStream: Enabling High-Performance Streaming between Data Centers . . . 151
12.3.1 Adaptive Cloud Batching . . . 151
12.3.2 System Architecture Overview . . . 153
12.4 Validation and Experimental Evaluation . . . 154
12.4.1 Accuracy of the Cloud Streaming Latency Model . . . 155
12.4.2 Individual vs. Batch-Based Event Transfers . . . 156
12.4.3 Adapting to Context Changes . . . 157
12.4.4 Benefits of Multi-Route Streaming . . . 158
12.4.5 Experimenting in a Real-Life Scientific Scenario . . . 158
12.5 Discussion . . . 160

13 Transfer as a Service: Towards Cost Effective Multi-Site Data Management **161**
13.1 Transfer as a Service . . . 163
13.2 Validation and Experimental Evaluation . . . 165
13.2.1 Evaluating the Inter-Site Transfer Options . . . 166
13.2.2 Dealing with Concurrency . . . 167
13.2.3 Inter-Site Transfers for Big Data . . . 168
13.3 Towards a "Data Transfer Market" for Greener Data Centers . . . 170

13.3.1 A Flexible Price Scheme for a Transfer Market 170
13.3.2 The Data Transfer Market 171
13.3.3 The Energy Efficiency of Data Transfers 173
13.3.4 Reliability 174
13.4 Discussion 174

Part IV — **Conclusions and Perspectives** **177**

14 Conclusions **179**
14.1 Achievements 179
14.2 Perspectives 182

Part V — **Appendix** **201**

15 Resumé **203**

Chapter 1

Introduction

Contents

1.1 Context 1
1.2 Contributions 2
1.3 Publications 7
1.4 Software 8
1.5 Organization of the Manuscript 9

1.1 Context

During the day when you are reading this, more data will be produced than the amount of information contained in all printed material in the world[1]. The Internet Data Center estimated the growth of data to be of a factor of 300 between 2005 and 2020, expecting to raise from 130 Exabytes to 20,000 Exabytes [64]. This Data Deluge revolutionizes both business, which now capitalizes the value searched in large data collections, and the process of scientific discovery, which moves towards a new paradigm: Data Science. Consequently, the applications need to scale and distribute their processing in order to handle overwhelming volumes, high acquisition velocities or great varieties of data. These challenges are associated to what is called "the Big Data phenomenon".

One factor which accelerated the revolution of Big Data and which emerged alongside with it, is cloud computing. The large, multi-site oriented infrastructure of clouds, which

[1]The amount of information contained in all printed material is estimated to be around 200 Petabytes [97], while IBM estimated that in 2012, in each day 2.5 Exabytes (1 Exabyte = 1024 Petabytes) of new data was created, and the amount continues to increase.

enables collocating computation and data, and the on-demand scaling provides an interesting option for supporting Big Data scenarios. Clouds bring to life the illusion of a (more-or-less) infinitely scalable infrastructure managed through a fully outsourced service that allows the users to avoid the overhead of buying and managing complex distributed hardware. Thus, users focus directly on extracting value, renting and scaling their services for a better resource utilization, according to the application's processing needs and geographical-distribution layout.

The typical cloud Big Data scenarios (e.g., MapReduce, workflows) require to partition and distribute processing across as many resources as possible, and potentially across multiple data centers. The need to distribute the processing geographically comes from multiple reasons, ranging from the size of the data (exceeding the capacities of a single site), to the distant locations of the data sources or to the nature of the analysis itself (crossing multiple service instances). Therefore, the major feature of such data-intensive computation on clouds is scalability, which translates to managing data in a highly distributed fashion. Whether the processing is performed in-site or across multiple data centers, the input needs to be shared across the parallel compute instances, which in turn need to share their (partial) results. To a great extent, the most difficult and compelling challenge is to achieve high-performance for managing the data at a large-scale, and thereby enable acceptable execution times for the overall Big Data processing.

The cloud technologies, now in operation, are relatively new and have not reached yet their full potential: many capabilities are still far from being exploited to a full degree. This particularly impacts data management which is rather far from meeting the more and more demanding performance requirements of the applications. High cost, low I/O throughput and high latency are some of their major issues. Clouds primarily provide data storage services, which are optimized for high availability and durability, while performance is not the primary goal. Some data functionalities, such as data sharing or geographical replication are supported only as a "side effect", while many others are missing: geographically distributed transfers, cost optimizations, differentiated quality of service, customizable trade-offs between cost and performance. All these suggest that data-intensive applications are often costly (time- and money-wise) or hard to structure because of difficulties and inefficiencies in data management in the cloud. In this landscape, providing diversified and efficient cloud data management services are key milestones for Big Data applications.

1.2 Contributions

Analyzing how clouds can become "Big Data - friendly", and what are the best options to provide data-oriented cloud services to address applications needs are the key goals of this thesis. Our objective is to provide data management solutions which enable high-performance processing at large-scale across the geographically distributed cloud infrastructures.Our work was mainly carried out in the context of the Microsoft Research-Inria Joint Center and involved collaborations with several Microsoft teams, within the framework of 2 projects between the KerData team and Microsoft. First, the A-Brain project enables scalable joint genetic and neuro-imagining analysis through large-scale computation on clouds. Second, the Z-CloudFlow project aims at enabling the execution of scientific workflows across multiple sites.

Contributions roadmap. The contributions are focused on performance aspects of managing data within and across cloud data centers. We started by addressing the challenges of executing scientific applications on clouds, with a particular focus on the MapReduce processing model. Next, we tackled the scalability aspects of single-site processing by extending the MapReduce computation across multi-site. These approaches were then applied to enable scientific discovery through large-scale computation, in the context of the A-Brain bio-informatics application. Processing Big Data at such a large-scale revealed several issues related to inter-site data management, that we addressed as follows. We proposed a scheme for dedicating compute nodes to data-related services. Next, we designed a transfer service architecture that enables configurable cost-performance optimizations for inter-site transfers. This transfer scheme is then leveraged in the context of real-time streaming across cloud data centers. Finally, we studied the viability of leveraging this data movement solution as a cloud-provided service, following a Transfer-as-a-Service paradigm based on a flexible pricing scheme. These contributions are summarized next.

Leveraging Locality for MapReduce and Workflow Processing on Clouds

As the cloud paradigm gets attractive for its "elasticity", enabling the computation units to access shared data concurrently and efficiently is critical. Using state-of-the-art cloud-provided storage is not feasible. In fact in current cloud architectures, the computational nodes are separated from the storage nodes and the communication between the two exhibits a prohibitively high latency. To address these issues, we propose a concurrency-optimized cloud data management solution, called TomusBlobs, at the platform level of the cloud usage. TomusBlobs builds on distributed storage solutions, which are used to federate the free local virtual disks of cloud nodes, to provide a globally-shared, data management platform for application processing. As demonstrated by the results, this approach increases the throughput more than twice over the cloud-provided storage by leveraging data locality. The benefits of the TomusBlobs approach were validated in the context of MapReduce, by building an Azure prototype, called TomusBlobs-MapReduce. It implements this computation paradigm and uses the proposed storage approach as a data management back-end. This solution reduces the timespan of executing scientific applications by up to 50 %. We further extended the TomusBlobs approach for general workflows, to leverage data locality for direct file sharing between compute nodes. To this end, we designed a file management system for federating the local disks of nodes for TomusBlobs. This approach exploits the workflow data access patterns to self-adapt and to select the most adequate transfer protocol, which speeds up data management by a factor of 2 over current data management options. This work involved the collaboration with Microsoft Research and a 3-month internship there. It led to 2 conference publications at CCGrid'12 and BigData'13.

Scaling MapReduce Processing across Geographically Distributed Sites

Another issue that we tackled is the scalability of computation on clouds. In practice, many Big Data applications are more resource-demanding, or operate on larger (and/or geographically distributed) data sets than typical cloud applications, thus requiring *multi-site processing*. Nevertheless, the existing computing models, such as MapReduce, are designed for single-cluster or single-site processing. To address this limitation and enable global scaling,

we proposed 2 extensions for the MapReduce paradigm. First, we proposed a two-tier hierarchical MapReduce scheme: the bottom tier distributes TomusBlobs-MapReduce instances across cloud data centers; the top tier computes the global final result of the processing. With this approach, we enabled efficient processing at a global scale, by disseminating the input and the computation tasks across all the available resources to aggregate compute power from multiple cloud sites. Second, we addressed the limitation of state-of-the-art MapReduce frameworks for reduce-intensive workloads regarding their lack of support for full reduction of the results (e.g., in a MapReduce process each reducer provides an output result). Therefore, we proposed Map-IterativeReduce as an extension of the MapReduce model, which enables to schedule efficiently the reduce jobs in parallel, based on a reduction tree, in order to compute a unique final result. Applying this technique in the context of multi-site MapReduce, enables to minimize the size and the number of expensive inter-site data exchanges, and thus to improve the overall data management. Using these extensions, we were able to achieve high scalability for scientific applications in public clouds, leveraging the processing power of 1000 cores across 3 geographically distributed data centers. Performance-wise, we were able to reduce the data management time by up to 60 % compared with state-of-the-art solutions. This work led to a publication in the MapReduce'12 workshop, held in conjunction with HPDC'12, and a journal publication in CCPE'13.

Enabling Scientific Discovery through Large-Scale Experiments on Clouds: The A-Brain Application Case Study

We applied the previous data management and processing approaches for enabling large-scale scientific discovery on clouds. As a case study, we worked in collaboration with a bio-informatics team from Inria Saclay, to run the A-Brain application, which aims to enable scalable joint neuro-imaging and genetics analysis. This pioneer technique would enable a better understanding of the variability between individuals and to explain brain diseases with genetic origins. However, both neuro-imaging and genetic-domain observations involve a huge amount of variables (i.e., in the order of millions). The high data and computation challenges prevented so far such complex analysis (conventional computation would take years, while the data space reaches petabytes order). Using our approach we ran this analysis on 1000 cores for 2 weeks across multiple Azure data centers, while consuming more than 200,000 compute hours – one of the largest scientific experimental setup on Azure up to date. Our tools enabled to provide the first statistical evidence of the heritability of functional signals in a failed-stop task in basal ganglia. This experiment demonstrates the potential and feasibility of our approach for supporting Big Data applications executions at large-scale by harnessing the available computation power from multiple cloud data centers. Additionally, this large, real-life experiment taught us important lessons, the most important being that the cloud model in which all types of data management exclusively rely on cloud-provided storage service becomes widely inefficient and obsolete. Therefore, it is critical to enable specialized cloud solutions for high-performance, geographically distributed data management. This project was presented through joint publications in Cloud Future workshop '12, the ERCIM electronic journal '13 and the Frontiers in Neuroinformatics journal 2014.

Dedicating Cloud Compute Nodes for Advanced Data Stewarding Services

The "data deluge" calls for scalable and reliable storage for cloud applications and a diversification of the associated data-related functionalities. Running large experiments reveals the limitation of the cloud-provided storage in coping with all the applications needs: it trades performance for durability and only provides basic put/get storage functions. However, executing Big Data applications requires more advanced functionalities such as logging mechanisms, compression or transfer capabilities. As an alternative to cloud-provided storage service, building such advanced functionalities in the application nodes, on top of a local collocated storage, can become rather intrusive and impact on the application performance. Therefore, we propose a different approach, called DataSteward, that combines the advantages of traditional cloud-provided storage (isolation of the data management from computation) with the ones of TomusBlobs (high-performance through the use of the local free disks of nodes). To this purpose, we dedicate a subset of the compute nodes and build on top of them a data management system. Thanks to the separation from the computation, this approach provides a higher degree of reliability while remaining non-intrusive. At the same time, applications perform efficient I/O operations as data is kept in the proximity of the compute nodes, by the use of a topology-aware clustering algorithm for the selection of the storage nodes. To capitalize on this separation further, we introduce a set of scientific data processing services on top of the storage layer that address the functionality needs of Big Data applications. Similarly to the concept of file from a traditional system, which is a generic object associated with a set of operations (e.g., move, view, edit, delete, compress), DataSteward confers a "cloud file" with its own set of actions. The evaluation results show that this approach improves by 3 to 4 times performance over cloud-provided storage. It can bring significant improvements for the management of applications data due to the topology-aware selection of storage nodes. The work was published in the TRUSTCOM/ISPA '13 conference.

High-Performance Inter-Site Data Sharing via Environment-Aware Multi-Route Transfers

One particular functionality that is missing in the current cloud data management ecosystem is the support for efficient geographically distributed transfers for applications. This is an important issue, as managing data across the geographically distributed data centers involves high and variable latencies among sites, and a high monetary cost. To address this problem, we introduce a cloud-based data transfer service which dynamically exploits the network parallelism of clouds via multi-route transfers, while offering predictable cost and time performance. The problem of low and unstable inter-connecting throughput between data centers is addressed through enhanced data management capabilities which adapt inside replication for faster data dissemination according to cost-performance constraints. Our system automatically builds and adapts performance models for the cloud infrastructure, in order to schedule the data transfers efficiently and to utilize the underlying resources effectively. The key idea is to predict I/O and transfer performance in order to judiciously decide how to perform transfer optimizations automatically over federated data centers. In terms of efficiency and usage, this approach provides the applications with the possibility to set a trade-off between cost and execution time. The transfer strategy is then optimized according to the trade-off. The results show that our approach is able to reduce the financial costs and

transfer time by up to 3 times. The work was published in the CCGrid'14 conference.

Enabling High-Performance Event Streaming across Cloud Data Centers

An increasing number of Big Data applications operate on multiple data centers around the globe and conform to a pattern in which data processing relies on streaming the data to a compute platform where similar operations are repeatedly applied to independent chunks of data. In fact, stream data processing is becoming one of the most significant class of applications in the world of Big Data, with vast amounts of stream data being generated and collected at increasing rates from multiple sources. Therefore, enabling fast data transfers across geographically distributed sites becomes particularly important also for applications which manage continuous streams of events in real-time. In order to adequately address the challenges of stream data processing across cloud sites, we performed an extensive performance evaluation study, in the context of CERN LHC processing experimental data. Our results indicate that current strategies for real-time communication in the cloud can significantly interfere with the computation and reduce the overall application performance. To address this issue, we propose a set of strategies for efficient transfers of events between cloud data centers and we introduce JetStream, which implements these strategies as a high-performance batch-based streaming middleware. JetStream is able to self-adapt to streaming conditions by modeling and monitoring a set of context parameters. The size of the batches and the decision on when to stream the events are controlled based on this model that characterizes the streaming latency in the context of clouds. To improve performance further, we aggregate inter-site bandwidth as we extend our previous approach for multi-route transfers across cloud nodes. The results show performance improvements of 250 times over individual event streaming and 25 % over static batch streaming, while multi-route streaming can further triple the transfer rate. The work led to 2 conference publications in CCGrid'14 and DEBS'14.

A Cost-Effective Model for Multi-Site Data Management

The global deployment of cloud data centers, brings forward new challenges related to the efficient data management across sites. The previous contributions focused on user-based solutions for tackling such challenges. Nevertheless, high throughput, low latencies, cost- or energy-related trade-offs are serious concerns also for cloud providers when it comes to handling data across data centers. Enriching the data-service ecosystem for delivering high-performance data management while reducing the costs and data center energy consumption is a key milestone for the business of cloud providers and for tomorrow's cloud data centers. To address these challenges, we propose a dedicated cloud data transfer service that supports large-scale data dissemination across geographically distributed sites, advocating for a Transfer as a Service (TaaS) paradigm. We argue that the adoption of such a TaaS approach brings several benefits for both users and cloud providers who propose it. For users of multi-site or federated clouds, our proposal is able to decrease the variability of transfers and to increase the throughput significantly compared to baseline user options, while benefiting from the well-known high availability of cloud-provided services. For cloud providers, such a service can decrease the energy consumption within a data center down to half compared to user-based transfer solutions. Finally, we propose a dynamic

cost model scheme for the service usage, which enables cloud providers to regulate and encourage data exchanges via a data transfer market. The work was published in the SRDS'14 conference.

1.3 Publications

Book Chapter

- **Radu Tudoran**, Alexandru Costan and Gabriel Antoniu. *Big Data Storage and Processing on Azure Clouds: Experiments at Scale and Lessons Learned*. In the book Cloud Computing for Data-Intensive Applications, to be published by Springer, 2015. Editors Xiaolin Li and Judy Qiu

Journal Articles

- Benoit Da Mota, **Radu Tudoran**, Alexandru Costan, Gaël Varoquaux, Goetz Brasche, Patricia Conrod, Herve Lemaitre, Tomas Paus, Marcella Rietschel, Vincent Frouin, Jean-Baptiste Poline, Gabriel Antoniu, Bertrand Thirion and IMAGEN Consortium. *Machine learning patterns for neuroimaging-genetic studies in the cloud*. In the journal of Frontiers in Neuroinformatics, Vol 8(31), 2014.

- Alexandru Costan, **Radu Tudoran**, Gabriel Antoniu and Goetz Brasche. *TomusBlobs: Scalable Data-intensive Processing on Azure Clouds*. In the journal of Concurrency and Computation Practice and Experience 2013

- Gabriel Antoniu, Alexandru Costan, Benoit Da Mota, Bertrand Thirion, **Radu Tudoran**. *A-Brain: Using the Cloud to Understand the Impact of Genetic Variability on the Brain*. ERCIM News, April 2012 - Electronic Journal.

International Conferences

- **Radu Tudoran**, Alexandru Costan and Gabriel Antoniu. *Transfer as a Service: Towards a Cost-Effective Model for Multi-Site Cloud Data Management*. In Proceedings of the 33rd IEEE Symposium on Reliable Distributed Systems (SRDS '14), Nara, Japan, October 2014.

- **Radu Tudoran**, Olivier Nano, Ivo Santos, Alexandru Costan, Hakan Soncu, Luc Bougé, and Gabriel Antoniu. *JetStream: enabling high-performance event streaming across cloud datacenters*. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS '14), Mumbai, India, May 2014, pp. 23–34. Acceptance rate 9 %.

- **Radu Tudoran**, Alexandru Costan, Rui Wang, Luc Bougé, Gabriel Antoniu. *Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers*. In Proceedings of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'2014), May 2014, Chicago, IL, US, pp. 92-101.Acceptance Rate 19 %.

- **Radu Tudoran**, Kate Keahey, Pierre Riteau, Sergey Panitkin and Gabriel Antoniu. *Evaluating Streaming Strategies for Event Processing across Infrastructure Clouds*. In Proceedings

of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid'2014), Chicago, IL, US, pp. 151-159, May 2014. Acceptance Rate 19 %.

- **Radu Tudoran**, Alexandru Costan, Ramin Rezai Rad, Goetz Brasche and Gabriel Antoniu. *Adaptive file management for scientific workflows on the Azure cloud*. In Proceedings of the IEEE International Conference on Big Data. Santa Clara, CA, US, October 2013, pp. 273-281. Acceptance Rate 17 %.

- **Radu Tudoran**, Alexandru Costan, and Gabriel Antoniu. *DataSteward: Using Dedicated Compute Nodes for Scalable Data Management on Public Clouds*. In Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TRUSTCOM/ISPA '13), Melbourne, Australia, June 2013, pp. 1057-1064.

- **Radu Tudoran**, Alexandru Costan, Gabriel Antoniu, and Hakan Soncu. *TomusBlobs: Towards Communication-Efficient Storage for MapReduce Applications in Azure*. In Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID '12), Ottawa, Canada, May 2012, pp. 427-434. Acceptance rate 26 %.

Workshops and Demos at International Conferences

- **Radu Tudoran**, Olivier Nano, Ivo Santos, Alexandru Costan, Hakan Soncu, Luc Bougé, and Gabriel Antoniu. *DEMO: Achieving high throughput for large scale event streaming across geographically distributed data-centers with JetStream*. In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS '14), Mumbai, India, May 2014.

- **Radu Tudoran**, Gabriel Antoniu, and Luc Bougé. *SAGE: Geo-Distributed Streaming Data Analysis in Clouds*. In Proceedings of the IEEE 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum (IPDPSW '13), Boston, MA, US, May 2013.

- **Radu Tudoran**, Alexandru Costan, and Gabriel Antoniu. *MapIterativeReduce: a framework for reduction-intensive data processing on Azure clouds*. In Proceedings of the 3rd international workshop on MapReduce and its Applications Date (MapReduce '12), in conjunction with HPDS 2012, Delft, The Nederland, June 2012.

- **Radu Tudoran**, Alexandru Costan, Benoit Da Mota, Gabriel Antoniu, Bertrand Thirion. *A-Brain: Using the Cloud to Understand the Impact of Genetic Variability on the Brain*. Cloud Futures Workshop, Berkeley, CA, US May 2012.

- **Radu Tudoran**, Alexandru Costan, Gabriel Antoniu, and Luc Bougé. *A performance evaluation of Azure and Nimbus clouds for scientific applications*. In Proceedings of the 2nd International Workshop on Cloud Computing Platforms (CloudCP '12), in conjunction with EuroSys, Bern, Switzerland, April 2012.

1.4 Software

TomusBlobs is a PaaS data management middleware that enables to federate the virtual disks of the cloud compute nodes into a uniform storage system. It provides several

features such as self-configuration, compute environment customization and policy-based role assignment. A version of the system is available as a storage backend to the Microsoft GenericWorker workflow engine [66, 153].
Size and language(s): ∼2K lines of code, C++ and C#

TomusBlobs-MapReduce is a MapReduce engine for the Azure cloud, built on to top of TomusBlobs. It is particularly optimized to support bio-informatics computation, including features such as full result reduction and bindings with various tools, libraries and scientific environments.
Size and language(s): ∼1K lines of code, C#, ASP.NET and Python

JetStream is a middleware solution for batch-based, high-performance streaming across cloud data centers. JetStream implements a set of context-aware strategies for optimizing batch-based streaming, being able to self-adapt to changing conditions. Additionally, the system provides multi-route streaming across cloud data centers for aggregating bandwidth by leveraging the network parallelism. It enables easy deployment across .Net frameworks and seamless binding with event processing engines such as StreamInsight.
Size and language(s): ∼2K lines of code, C#

Cloud Benchmark Service is a cloud web service provided as a Software as a Service, that enables to benchmark online the data stage-in performance of the cloud data centers. Building on several web technologies it provides both an online web-browser interface and a a console-based API.
Size and language(s): ∼2K lines of code, C++, C#, ASP.NET MVC, AJAX and JavaScript

1.5 Organization of the Manuscript

The rest of the manuscript is organized in 4 parts.

The first part discusses the general context of this thesis and presents the corresponding state-of-the-art. First, we introduce in Chapters 2 and 3 two paradigm shifts: Big Data and Cloud Computing and describe how these are changing the way in which data is managed. Next, in Chapter 4 we present the main challenges that arise for supporting such Big Data processing on cloud platforms. Finally, in Chapter 5, we give a global overview of the current research efforts and approaches on this subject.

The second part includes the next 4 chapters (Chapters 6 to 9) and presents the first part of our contributions. We mainly focus in this part on data management aspects of *single site processing*, while in the next part we consider the general case of *multi-site processing*. The goal is to understand what are the best options to support efficient scientific Big Data processing on the cloud. To this end, we use a real-life application from bio-informatics, A-Brain, with high computation and data requirements which are described in Chapter 6. Next, in Chapter 7 we introduce the TomusBlobs data management approach, which we designed for supporting efficient, single-site processing of such scientific applications. Moreover, we propose a MapReduce solution on top of TomusBlobs, which we extend in Chapter 8 to a hierarchical

multi-site MapReduce to provide global scalability. Based on these approaches, we enable the execution of the A-Brain joint genetics and neuro-imaging data analysis through a large scale, long-running experiment. The scientific results enabled by our solutions as well as the lessons learned from this experiment are described in Chapter 9. The most important lesson is that there is a critical need, in the context of Big Data, for tools that would provide high-performance data management across data centers, which is the focus of the next part.

The third part includes Chapters 10 to 13, and presents several contributions, addressing the issues identified in the first part for Big Data scientific applications, that improve the performance of managing data across multiple cloud data centers. First, we introduce in Chapter 10 a method for dedicating compute nodes to a data management system, on top of which we propose a set of data services to enrich the support for Big Data applications on clouds. In Chapter 11, we propose a service that optimizes cost and performance for data exchanges across cloud sites, by aggregating inter-site bandwidth through multi-route transfers over compute nodes and data centers. Next, we leverage this transfer scheme in 2 contexts. First, in Chapter 12 we address the lack of cloud services for fast real-time communication and propose JetStream, a high-performance, batch-based streaming solution. Finally, in Chapter 13 we explore architectural options to migrate this transfer scheme into a cloud-provided Transfer as a Service, with a flexible cost and a data transfer market.

Finally, Chapter 14 concludes this work and presents the contributions and the perspectives brought by our solutions.

Part I

Context: The Landscape of Big Data Management on Clouds

Chapter 2

Background: The Era of Big Data

Contents

2.1 The Data Deluge 13
2.2 Data Science: The Emergence of a New Scientific Paradigm 16
2.3 Discussion 18

Much of today's and tomorrow's data, captured by the growing networks of sensors or generated by large simulations and computational models, are likely to reside forever. Substantial efforts are being made to make such collections of data, particularity the scientific ones, publicly accessible for the purpose of continuous analysis. Gordon Bell synthesizes this by stating that he believes «*that we will soon see a time when data will live forever as archival media — just like paper-based storage — and be publicly accessible in the "cloud" to humans and machines*» [89]. Acquiring, permanently storing, curating and processing the exponentially growing volumes of data is refereed to as *Big Data*. Performing data-intensive computation on Big Data sets to enable cutting edge discoveries is now considered as the basis for a new, *fourth paradigm* for science.

2.1 The Data Deluge

Data Deluge is becoming a reality. Internet Data Center (IDC) estimated the growth of data to be of a factor of 300 between 2005 and 2020, expecting to raise from 130 Exabytes to 20,000 Exabytes [64]. Keeping up with such a pace, by transferring, storing, organizing and processing the data, stands as a difficult challenge, generically refereed to as Big Data. Even so, the data deluge is transforming both the business and the scientific domains: business mines data for patterns which can be capitalized, while science moves towards a new paradigm where data-intensive computation drives discoveries. The data-oriented quest and the re-

lated Big Data challenges raise the requirements in terms of scale and performance expected from the next data management systems.

The Big Data concept refers to the exponential growth of the data volumes [25], the actual processing of the large volumes of data [95] or more generically to all the computing phases required to extract useful information [64, 184]. One can observe that several definitions coexist and are used and accepted depending on the community. However, in order to avoid potential confusions and to characterize better the concept of Big Data, a set of attributes were identified as defining characteristics. This set of attributes is referred to as the *V's* of Big Data, according to their common name initial. Considering that the Big Data ecosystem is highly dynamic, the set is expanding to include new V's (i.e., challenges) that are continuously identified. Next, we present the original set of the 3 V's as introduced in [115] and 2 additional ones which are widely accepted and used. Different aspects of these challenges are presented also in [51, 184].

Volume

It represents perhaps the main feature that one associates with the concept of Big Data. This association with the magnitude of the data set arises naturally as all domains tend currently to collect and store massive amounts of data.This behavior is encouraged both by the low costs to store data and because having models which result from large data set tends to provide more accurate results, from the data analytics point of view. In fact, the size dimension of Big Data represents the primary challenge to the existing data management systems. Accommodating the growing volumes calls for scalable storage solutions and distributed processing engines. Furthermore, the size of the data can become big enough such that it has to be stored across multiple data centers, which requires high-performance solutions capable to operate in a geographically distributed manner. An illustrative example is the CERN LHC Atlas experiment [13], which generates 40 PB of data per year, which is disseminated across the storage of tens of collaborative institutions across the globe. Analyzing this geographically distributed data set, even by means of incremental processing, overpasses the capacity of local scientific infrastructure. This was the case for the Higgs boson discovery, in which the computation was also extended to the Google cloud infrastructure [43]. Hence, processing across geographically distributed data centers and inter-site data management become a necessity in the Big Data era, due to the challenges raised by the data volumes.

Velocity

The high rates at which data are collected by organizations or flows into the processing engines make data velocity to gain importance alongside with volume. The common terminology used for fast-moving data is "streaming data". Initially, the velocity-related challenges were restricted to specific segments of industry, but it becomes a problem of a much broader setting with the Internet of Things. Financial tickers, stock market analysis, monitoring systems of large web services, network of sensors for wide-areas or scientific observatories are all concerned with the speed at which data is collected, streamed and processed in real-time [68, 110]. In fact, it is expected that in the next years most part of Big Data will be collected in real-time, which means that the speed to collect data over-passes the rate to

artificially-produce them [172]. Real-time data processing (i.e., stream processing) is necessary due to various reasons. For example, keeping the storage requirements practical can require pre-processing to filter out the useless parts in scenarios with high data rates. Additionally, many scenarios require the information to be extracted from data immediately or within a maximal (short) delay. However, depending on the locations of the data sources which produce the data, this task of real-time processing can be particularly challenging. The data sources can be geographically distant from the stream processing engine, which adds also the problem of latency to the challenges related to the streaming rate. An illustrative example of such a scenario is the Ocean Observatory Initiative [23, 137] in which the collected events are streamed to Nimbus [12] clouds. Hence, supporting the paradigm shift brought by Big Data calls for data management solutions which consider and address not only the size aspects of data, but also the challenges raised by streaming, all the more in geographically distributed setups.

Variety

Collecting data from a variety of sources leads to a high heterogeneity. In fact, dealing with Big Data sets most often implies handling data without a predefined relational structure. Therefore, curating the data before storing and processing them becomes a critical tasks and a challenge on its own. However, pre-processing and determining a relational scheme before storing it is a complex task considering the large volumes. In the few cases when this phase is possible, the scalability limits of traditional databases [89, 162] can still arise as a prohibiting factor to address the variety aspect of Big Data via relational schemes. As a consequence, the common approach is to handle data in an unstructured fashion, e.g., storing data in large binary objects. On the one hand, such an approach provides scaling and performance advantages. On the other hand it amplifies the variety problem of Big Data sets, as most of the contextual and self-describing information is lost. As a result, the process of extracting (co)relations and ordering the data is coupled with the data mining itself, sometimes becoming the computation itself. Finally, with the emergence of data-intensive science, as discussed in the next section, multi-disciplinary collaborations grow in number and importance. A key direction enabled by such joint efforts is trying to correlate the heterogeneous data sets of the distinct disciplines in order to discover new ways of explaining the life- or universe-related observations. Examples of such efforts range from combining genetic sets with medical imagery [146] to merging climate and astronomical models with cartographic data [126, 179]. Such ambitious goals call for efficient, large-scale tools which can handle the variety aspects of Big Data.

Veracity

The trustworthiness of data impacts theirs value. Therefore, one of the newly identified challenge when dealing with Big Data is veracity, which generically synthesizes the correctness, quality and accuracy of the data. The concerns related to the veracity apply both to the input as well as to the result harvested when mining it. Veracity becomes an important aspect due to the diversity of the sources and forms that Big Data takes, which provides less control over its correctness. Malfunctioning sensors, typos in social media feeds, colloquial discourses in news media, systematic errors and heterogeneity of measuring devices, all

need to be accounted for during the analysis phase. Sometimes, the volume can compensate for the lack of accuracy. But tackling veracity via volume needs to instrument the analytic models properly, which is achieved most often at the expense of extra computation. Therefore, providing tools which are able to scale the computation in order to ensure high and customizable confidence levels for the analysis is critical for the development of Big Data business and data-intensive sciences. Moreover, accommodating provenance information and trust levels for the input and results in the management process [49, 60] are some of the key points required to increase the quality and value of data.

Value

Collecting, storing and analyzing Big Data is useless unless it produces value. Therefore, this aspect goes alongside and determines any previous or future challenge of Big Data. It can be safely stated that "Value" is the ultimate goal of Big Data, being the driven factor of this technological revolution. Dealing with Big Data is a complex task and it involves significant costs. Therefore, the benefits gained, whether financial or scientific, must compensate the resources and efforts which are invested. This observation raises a very important aspect to be considered when dealing with Big Data: the efficiency trade-off between cost and performance. The new management solutions need to provide mechanisms for estimating both the cost and performance of operations (e.g., for streaming, storing, processing, etc.). Based on these estimations the users can then choose the cost they are willing to pay for a certain performance level. In turn, the management systems need to optimize their resource usage according to the specified criterion to meet the budget/performance constraints. Designing such customizable trade-off mechanisms is needed because the density of the value in the data sets is not uniform nor identical among different applications. This shows that the value aspect of Big Data, needs to be considered not only from the point of view of the worthiness (or profitability) of the analysis, but also as a designing principle of the management and processing frameworks.

2.2 Data Science: The Emergence of a New Scientific Paradigm

As Data Deluge is becoming a reality, Data Science is emerging, as the new, fourth paradigm of science [72, 89]. This paradigm shift happened naturally. Centuries ago, science was mainly done through empirical observations. The next step was to synthesize those observations about the world in theoretical models. When those models became too complex to be solved and interpreted analytically and when technology allowed it, science moved towards a computation paradigm, using computers to analyze and simulate the theoretical models. However, this computation-driven science led to a continuous growth of the scientific data sets. This growth trend was also accelerated by the increase in efficiency and diversity of the tools used to collect and deliver the data to the scientists. As a result the scientists shifted their focus in the last years, searching for discoveries in their large scientific data sets. The techniques and technologies which are developed to make this "search" more efficient, *"distinguish this data-intensive science from computational science as a new, fourth paradigm for scientific exploration"* [24].

In the last years, data-intensive science encouraged the development of several large sci-

entific projects. Collecting, curating and computing data, the 3 phases which constitute the data science, face complex challenges due to the massive quantities of data produced [72]. For example, the Pan-STARRS [142] project consists of an array of celestial telescopes which monitor the Solar System to detect potentially hazardous objects or threats, such as asteroids. The project is designed to collect images of a resolution of 1.4 Gigapixels, producing 2.5 Petabytes of data per year. This type of projects, which are a direct consequence of, and illustrate perfectly, the fourth paradigm of science, are referred to as virtual observatories [89, 90]. Other examples of virtual observatories are the Australian Square Kilometre Array of radio telescopes project [156] or the Ocean Observatory Initiative [137]. Such observatories can monitor wide and remote areas. Thus, the data collection is particularly challenging, as it involves real-time geographical data streaming. The Large Hadron Collider [38] is a physics project based on a collaborative multi-institutional effort. It produces and collects about 40 Petabytes of data each year, aiming to explain the early stages of the universe. Its most recent and notorious discovery was the detection of the Higgs Boson in the Atlas experiment [13, 43]. The data produced is disseminated, stored and processed across more than 80 computing centers worldwide, federating and linking 100,000 CPUs [37, 143]. Finally, genetic studies are another example of an area which benefited from the emergence of data-intensive computing. Gene sequencing is the process of coding genome regions with the purpose of analyzing and searching for clues about diseases. It used to be an expensive and slow process. However, the US X prize allocated recently 10 million for genomics [180]. This enabled to increase the amount of sequenced genes from 25 thousands to 3 million. In turn, this opened the door for a Big Data approach for genetics-based studies, which can now search for the disease roots or for ways to explain the variability between individuals by studying genomic correlations.

This data deluge is supported by the capacity of the networks to scale within and across infrastructures. With such growing sizes of data collections beyond the Petascale limit, one data center, let alone one cluster or multi-core machine, can no longer store nor process entire data sets in reasonable time. Scale-out solutions are then the right approach: partitioning data into small blocks and disseminating it over the network across the locally attached storage of nodes, enables one to manage these large data volumes [158]. The drawback which comes with this approach is that data gets heavily partitioned, which prevents further usage of traditional database systems. Additionally, databases tend to reach also a volume limit in the order of tens of terabytes [89, 142, 162]. The alternative solution, which gains in popularity today, for addressing both the volume and the partitioning challenges, is to deploy and scale a simple data-crawling, shared-nothing strategy over the partitions – the MapReduce paradigm [46]. The solution is less powerful than the index mechanisms that databases offer [144], but becomes a default Big Data tool due to its massive scalability capabilities. We further discuss the MapReduce approach in Chapter 4.3.

In this data-intensive context, with the partition of the growing data sets, data delivery is hitting a wall. In the last years, data and storage grew exponentially (increasing 1000 times) while the access speed to data improved only 10 times [89]. Another critical issue is that current data management tools cannot handle the Petabyte scale. Accessing via FTP a Petabyte of data is clearly not feasible; the solution to accomplish such transfers requires at least a parallel access scheme [72]. Moreover, moving at least part of the computation to the data rather than vice-versa can bring significant benefits. Many other such optimizations can and need to be provided in order to support the Petabyte scale and beyond. The main problem is that

scientists do not have currently off-the-shelf solutions to manage data in their Big Data application scenarios [73, 89]. This lack of solutions and support for data management and the inherited low I/O performance drastically limits scientific observation and discovery process. In fact, the performance aspects of handling Big Data are the most critical challenges that need to addressed, being one of the primary motivating factors of this thesis.

2.3 Discussion

Big Data is reshaping the way in which the scientific discovery is, and will be, done. Physics, biology, climate, astrology and many other domains are now searching for scientific answers in data collections. Nevertheless, this paradigm shift raises new challenges, more complex than previously dealt with. Traditional compute and storage systems reach their limits, as they cannot handle the volumes, velocities or varieties brought by Big Data. New ideas, approaches and technologies are needed, alongside with powerful compute platforms.

One factor which encouraged the Big Data revolution and which emerged alongside with it is cloud computing (discussed next, in Chapter 3). The large, multi-site oriented infrastructure of clouds, which enables collocating computation and data and on-demand scaling, represents an interesting option for addressing the challenges brought by Big Data. However, its role for the data-intensive sciences is not fully established [89]. On the one hand, migrating the scientific applications in the cloud is encouraged by the benefits they bring for the web-base applications and businesses for which they were intended. On the other hand, it is not clear how they can support various scientific scenarios with potentially contradicting goals and requirements, such as low latency, resources scaling, high bandwidth, data migration and partitioning, etc. Analyzing how clouds can emerge to become *"Big Data - friendly"* for science, and what are the best options for them to provide data-oriented services to address applications needs are the key goals of this thesis.

Chapter 3

Background: Cloud Computing

Contents

3.1 Overview 19
3.2 The Cloud Computing Paradigm 21
3.3 The Cloud Storage Paradigm 23
3.4 Discussion 25

Cloud computing emerged as a new paradigm for providing diversified resources, such as computation power, storage capacity and network bandwidth. These resources are offered to users as services, at different levels of granularity, by means of virtualization. Users lease services *on-demand* and elastically scale them, on short notice, according to their needs, on a *pay-as-you-go* basis. This paradigm revolutionized the way in which we approach resource provisioning, moving the ownership and management of resources from in-house premises to the cloud provider. Cloud computing is proving fundamental to the construction of systems capable of rapid scaling and of delivering high reliability. This technological evolution is particularly impacting in the Big Data era, where an increasing number of applications need to scale to meet their compute and storage challenges.

3.1 Overview

Cloud computing allows users to focus on extracting value, renting and scaling the services for an optimal resource utilization [169]. In turn, cloud vendors manage the resources and offer different levels of control over them. With respect to the providers and their accessibility, clouds are public (i.e., offered by vendors openly) or private (i.e., with a restricted access). A taxonomy based on usage levels can be determined, depending on what falls under the responsibility of the provider to administrate, as shown in Figure 3.1. The resulting categories are labeled *"as a Service"*, and provide the following functionality.

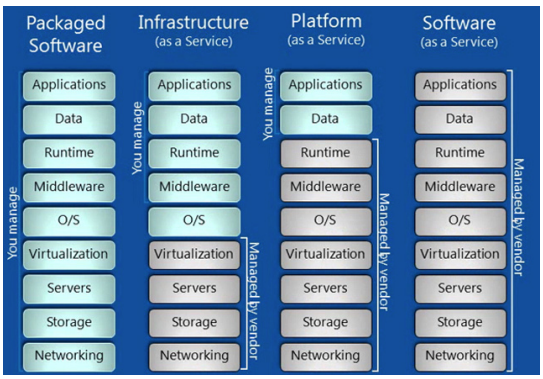


Figure 3.1: The cloud usage models [171]

Figure 3.2: The global presence of a cloud data centers exemplified for the Microsoft Azure cloud [16]

Infrastructure as a Service — IaaS. Provides resources with minimum administrative support by the cloud vendor, being in the most raw form (i.e., close to the bare hardware) the virtualization techniques allow it. Cloud vendors split their large storage and compute capacity, and build ad-hoc virtual environments as demanded by users [169]. The typical way of exposing the compute resource to users is through virtual machines (VM), in which the software stack that runs the user services is deployed. Any functionality that needs to be built falls under the responsibility of the users for installing, configuring and managing.The most illustrative example of a IaaS cloud is Amazon Elastic Compute Cloud (EC2) [9]. Alternatively, several open source toolkits were developed which can virtualize (typically private) physical resources as a IaaS cloud, for instance Nimbus [134], OpenNebula [138], Eucalyptus [54] and OpenStack[139].

Platform as a Service – PaaS. Offers an alternative to exploit the cloud resources, which are presented as a uniform software platform rather than as a virtualized infrastructure. Much of the responsibility for the deployment, configuration, management, fault tolerance and recovery mechanisms of the infrastructure is taken by the cloud provider. Users focus on the application logic being transparently provided with the runtime environment of their choice (e.g., .Net, Java SDK, databases, etc.) or even complete programming models such as MapReduce [46], Dryad [100] or Azure Roles [125]. The well-known examples of PaaS cloud offerings are the Google Apps Engine [71] and Microsoft Azure [125].

Software as a Service — SaaS. Builds on the cloud resources to deliver a high-level service, typically with a specific functionality. It stands as an alternative for cloud vendors to leasing their infrastructures. They can host their own services and provide online, most commonly through web-browser interfaces, various functionalities to users, such as: office tools, content delivery network (CDN), collaborative tools, multimedia or web-mail hosting. The main advantage of this model is that it frees users from any kind of management, software installing, updates or patches, as they simply consume the service. Some of the representative examples are the Google Docs and Microsoft Office 365 office tools or the e-commerce services provided by saleforce.com.

Anything as a Service — XaaS - Due to the success of this taxonomy model, and the popu-

larity gained by clouds, many other paradigms and functionalities tend to be labeled and provided *"as a Service"*. Examples range from Database (or simply Data) as a Service [93, 125], Network as a Service, Secure Logging as a Service [183], Business Integration as a Service [178], Search as a Service [154], etc. This "etc." is in fact what defines this class, saying that anything/everything can ultimately be provided as a service. Clearly, this leads to a thin border between categories (as it is the case for the main ones), since much of these examples can be seen as SaaS, PaaS or even IaaS. In fact, labeling something *as a Service* is merely a differentiation and an acknowledgment of its high value, which is offered online to users rather than exploited in-house. Most of the XaaS utilities serve as alternatives to users for building and running their own applications. The fact that users and developers are able to extend the cloud computing paradigm, build, then offer such services, according to application needs, is one of the key advantages that drive the paradigm shift introduced by cloud computing. This is, as well, the model that we adopt for providing and delivering our contributions, such as Monitor as a Service, Streaming as a Service, or Transfer as a Service. These approaches are discussed in the following chapters of this thesis.

The cloud infrastructure is composed of multiple, large data centers, which are geographically-distributed around the globe. However, the interconnecting infrastructure between the sites is not the property of the cloud provider as the bandwidth is leased from the Internet Service Providers (i.e., Tier 1 ISP). Nevertheless, such a geographically distributed architecture enables cloud users to rent resources in the proximity of their users or data sources. Figure 3.2 illustrates this multi-site concept with the Microsoft Azure cloud. A cloud data center offers both computation power and storage capacity, which are virtualized and offered as services to customers [34, 74]. Collocating data and computation within the site is a pragmatic choice as it enables applications to reduce the otherwise high costs of moving data across wide areas. In what follows, we present an overview of these two aspects.

3.2 The Cloud Computing Paradigm

The Data Centers. The *data center (or the site)* is the largest building block of the cloud. It contains a broad number of *compute nodes*, which provide the computation power, available for renting. The typical pricing model is computed at an hourly rate, based on the computation power of the resources acquired by the user. The data center is organized in a hierarchy of switches, which interconnect the compute nodes and the data center itself with the outside world, through the Tier 1 ISP. Multiple output endpoints (i.e., through Tier 2 Switches) are used to connect the data center to the Internet [74]. For the sake of fault tolerance, the overall infrastructure within a data center is partitioned with respect to the switches and racks. These partitions are referred to as *fault domains*. The users services are distributed across several such fault domains. Finally, the compute resources are managed by hypervisors, which deploy, stop or update user services. They are also in charge of monitoring the infrastructure.

The Compute Nodes. Clouds virtualize the *compute nodes* and lease their compute power in different flavors, in the form of virtual machines (VMs). At PaaS level, all VM instances

assume the same role, while at IaaS they have no role. Hence, in both cases, achieving self-configuring for the services deployed across the VMs is critical. The physical nodes are exploited in a *multi-tenant* manner, being shared by several VMs, generally belonging to different users. Depending on the amount of physical resources they hold VMs range from tiny instances which share the CPU, to larger ones which fully allocate anything between 1 and the maximal number of CPUs of a physical machine (i.e., 8, 16 or 32 depending on the provider and resource type). The memory, local storage and the network quota tend to increase proportionally for the larger VM types (e.g., an Extra Large VM has 8x more resources than a Small one, while Large or Medium VMs have 4x, respectively 2x more). Despite having CPUs mapped in a 1 to 1 relation with the virtual CPUs, the only option that frees a users from multi-tenancy (i.e., physical machine shared by VMs of multiple users) at compute node level is to use the largest VM types. As such VM types fully occupy the physical nodes (e.g., Extra Large VMs for Azure cloud) no other users VMs will be run on the physical machine. Nevertheless, even with the largest VM type, the rest of the resources in the data center, such as network switches or links, are shared among all users.

The Deployment. The VMs that host a user application form a virtual space, also referred to as *deployment*. The VMs are placed on different compute nodes in separate fault domains in order to ensure fault tolerance and availability for the service. Typically a load balancer distributes all external requests among the VMs. The topology of the VMs in the data center is unknown to the user. Several other aspects are invisible (or transparent) to users due to virtualization, e.g., the mapping of the IPs to physical node, the vicinity of VMs with each other, the load introduced by the neighbor VMs, if any, on the physical node, etc. A deployment is limited to a single site, and depending on commercial constraints and priorities, it can be limited in size. This implies to use several independent deployments if an application needs to be executed across several sites, or across a very large number (e.g., thousands) of CPUs. These are virtually isolated one from the other, e.g., the virtual private network from one is not accessible by the other. Hence, aggregating their functionality requires an actual service orchestration similar with creating a workflow from web-services. This raises a difficult challenge for managing the data uniformly across these different virtual spaces, potentially located in geographically distant sites.

Examples of major providers of cloud computing services are the following.

Amazon Elastic Cloud Computing (EC2) [9] are the computing services traditionally offered at IaaS level, by Amazon. Being among the first vendors which rented their compute capacity to users, it is often regarded as the reference cloud service, with many of their APIs adopted as "de-facto" standards. The virtual machines, called instances, provide users with complete control over the software stack. In the recent years, the Amazon cloud services evolved and diversified their offering to include data-oriented analytic platforms such as MapReduce.

Microsoft Azure [125] is a cloud platform that offers PaaS services, and more recently also IaaS ones. At IaaS levels, the nodes are offered as Virtual Machines, while at PaaS level these VMs take roles, i.e., Web Roles and Worker Roles. The VMs run either Windows or Linux-based OS and can provide different run-time software stacks such as .Net or Java. As Amazon, the cloud framework introduced recently a set of business-oriented

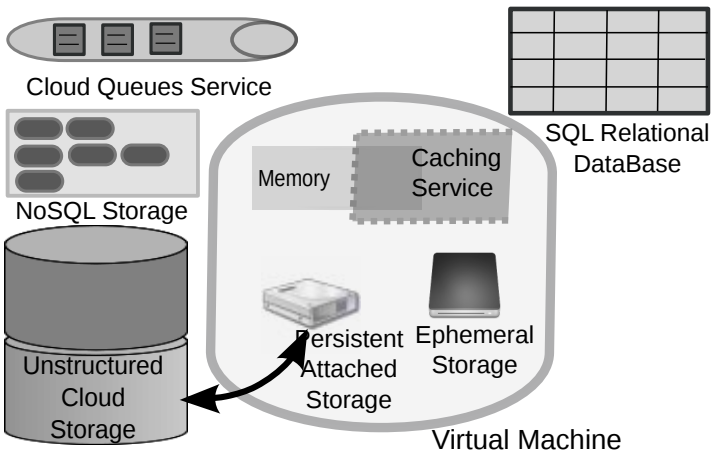


Figure 3.3: An overview of the cloud data services

services, enabling users to run MapReduce jobs (further discussed in Chapter 4.3), develop applications or orchestrate business or scientific processes.

Google App Engine [71] is the PaaS computing cloud from Google. It offers users the possibility to develop applications in several programming languages (e.g., Python, Java, PHP, Go) and to integrate several other code extensions or technologies (e.g., Node.js, C++). The infrastructure is optimized for high networking performance and fast provisioning for applications scaling. Regarding the pricing model, Google App Engine offers a fine grain model, charging VMs usage by the minute, after the first 10 minutes, unlike Amazon and Azure clouds which charge by the hour.

3.3 The Cloud Storage Paradigm

Clouds offer also storage capacity for data management, alongside with compute power. This is either virtualized, or delivered to customers via services, typically with *put/get* REST APIs [9, 35]. The pricing model is based on the data size, time frame and number of operations to the data. The cloud data management ecosystem consists of several services as depicted in Figure 3.3. These are geared for various types of data and maximize a different (typically conflicting) set of constraints. Next, we present an overview of these services.

Ephemeral Storage. In addition to computing power, the VM instances also offer storage capacity, in the form of virtual disks. This storage capacity can reach several TBs (e.g., 2 TB for an Extra Large VM in Azure) depending on the instance type. However, the data stored in the virtual disks are ephemeral, as they persist only for the lifetime of the instance, being subject to losses caused by failures of the instance. The storage is free of charge, in the sense that it comes with the cost of leasing an instance. Considering the low latency between these virtual disks and the compute resources, thus leveraging data locality, ephemeral storage is an interesting option for hosting and buffering data for high-performance processing.

Unstructured Cloud Storage. This represents the standard cloud offering for sharing application data, which is available both to cloud compute nodes and to on-premise ap-

plications. These cloud storage services are distributed storage systems deployed on storage nodes and running complex protocols for partitioning, replication and consistency to ensure that data is always available and durable, even when machines, disks and networks fail.A strong consistency protocol is used for the replicas within the site, while concurrent updates are handled through the use of timestamps and by applying a "first commit wins" strategy [35]. This makes the these cloud storage services good candidates for persistently storing input/output data. Azure Blobs is the main storage service of the Azure cloud [125], S3 is the equivalent service from Amazon cloud, while Google offers Google Cloud Storage [71]. Users can create several storage accounts for a cloud storage subscription, each bounded to a geographic location (i.e., a data center), but with the possibility of data redundancy across sites, for disaster recovery. Data is stored as binary objects (BLOBs) and usually organized in a two-level namespace structure (i.e., blobs and containers). These BLOBs tend to have an upper size limit for the amount of data stored in the order of terabyte. Despite collocating them within the site with the compute deployments, the high-latency REST interfaces make them inadequate for supporting high-performance data management, which is understandable as they are optimized for data availability and durability.

SQL Relational Databases. In the last decades, relational databases have became the primary option for storing application data, leveraging their support for indexed storage and ACID guarantees (atomicity, consistency, isolation and durability). Therefore, cloud vendors provide such services (i.e., Database as a Service), offering online relational databases on top of their storage capacity. Nevertheless, even with the cloud-deployed databases, the storage limitations of this technology persist. For example, the Azure SQL database service offered by the Microsoft cloud offers a maximum storage space of 150 GBs per user subscription.

NoSQL Storage for Structured Data. With the emergence of Big Data, the storage limits that come with the ACID guarantees in relational databases, become a major bottleneck. To address this issue, the NoSQL systems weaken these properties and propose instead to store and structure data in the form of type-value or key-value pairs. Examples of such systems are Cassandra [113], Google Bigtable [40] or Yahoo PNUTS [44]. Due to their efficiency and performance to store large data sets, such systems were also ported on the clouds (e.g., Azure Tables [35] or Amazon Dynamo [47]). Unlike databases, they do not necessarily enforce a fixed schema enabling a various number of properties or types, within or across rows. As a result of porting such services on the clouds, users have an alternative to the unstructured cloud storage, which enables them to structurally store and partition (i.e., similarly with the indexes in relational databases) terabytes of data (e.g., Azure Tables offer a up to 100 TB per account).

Cloud Queue Service. One of the data management services provided by clouds is dedicated for nodes synchronization.To this purpose, nodes can exchange small (e.g., 64 KB), short-term (e.g., 1 week) messages via the cloud queue service. A typical usage scenario consists of compute nodes, acting as front-ends, enqueueing jobs for the nodes acting as workers, as in a master - worker model. Cloud Queues are designed as a communication mechanism capable to support a high number of concurrent operations from many clients and to allow thousands of message transactions per second. Examples of such services are: the Azure Queue [125] or Amazon Simple Queue Ser-

vice (SQS) [9]. An interesting fault tolerance property offered by Azure queue is that a message which is dequeued, is not immediately deleted from the service, but hidden for a certain period of time. The message is then either explicitly deleted, marking its successful processing, or becomes visible again after the hidden period expires. In this way, the service guarantees that each message is processed at least once. However, the hiding period is limited to hours, which creates a management overhead for long running tasks. It can be the case for scientific Big Data applications, for which users need to provide additional mechanisms that support longer compute intervals.

Persistent attached storage. The virtual attached disks of a VM have an ephemeral lifetime, while the persistent cloud storage is accessible via REST interfaces, which in turn might require changes in application semantics. To address this issue, cloud vendors provide a middle-ground solution, in the form of persistent attached storage, which enables to mount BLOBs in a VM and expose them as a local disk. In this way, applications manage files, while the lifetime of these files is independent of the instance that mounts the BLOB. Although a BLOB can be mounted only to a single compute instance at a given moment, it can be reused any number of times and by other instances. Examples of persistent attached storage are Amazon Elastic Block Storage (EBS) [7] or Azure drives [17].

Caching Service. Depending on their type, the compute instances in a deployment can offer a significant memory capacity. One option to exploit this, is to build a caching service across the memory of the compute nodes, in order to enable fast, in-memory storage. Cloud providers offer such tools, for example Microsoft offers Azure Caching [15], which can be deployed either on a set of dedicated compute nodes, or across all the leased compute nodes within a deployment. For the latter case, each VM will have a portion of its memory dedicated to this service. Performance is high due to data locality and in-memory data access. However, cache misses can be quite frequent, especially when the memory capacity is reached, and the data needs to be brought from the cloud storage where it is checkpointed for persistence.

3.4 Discussion

The cloud computing model enabled a major paradigm shift for online applications. Clouds bring to life the illusion of a (more-or-less) infinitely scalable infrastructure managed through a fully outsourced service.Instead of having to buy and manage hardware, users rent resources on-demand. However, cloud technologies have not yet reached their full potential. With the emergence of Big Data applications, it is critical to leverage the cloud infrastructures to cope with the challenging scale (e.g., volume, velocity) of the Big Data vision. In particular, the capabilities available now for data storage and processing are still rudimentary and rather far from meeting the more and more demanding applications requirements. Hence, porting data-intensive applications to the clouds is challenging and brings forward many issues, which we discuss in the next chapter.

Chapter 4

Objective: Processing Big Data on Clouds

Contents

4.1 Overview of Big Data Applications 27
4.2 Challenges and Issues 28
4.3 Big Data Processing Models: MapReduce and Beyond 31
4.4 Discussion 32

This thesis focuses on the Big Data applications, particularly the scientific ones. Commonly, large scientific applications such as climate modeling, simulations of the evolution of diseases and viruses, stream flows analysis, etc. were executed on specialized and powerful, yet expensive, infrastructures — the supercomputers [29]. However, not all research groups have access to build and manage such systems. This raises a barrier for the development of science. Fortunately, the cloud computing paradigm democratizes science, by facilitating the access to large-scale resources to any users/researchers without any prerequisites and with a minimal financial effort. To this purpose, we are interested to investigate the main requirements and challenges for enabling efficient processing of such Big Data applications on clouds.

4.1 Overview of Big Data Applications

The existing cloud infrastructure and the related services can efficiently and effectively support small applications with moderate resource requirements. Examples range from national traffic simulations (e.g., train scheduling [70]) to some bio-medical computations [76]. However, clouds reach their limitations when handling Big Data applications that require a large

number of resources, rapid scaling or high-performance computing [67, 93, 149, 150]. For example, a request for a large number of compute resources at once is not always successfully fulfilled [102].Another issue is that applications can obtain a lower and variable CPU speed than the one requested when choosing the VM type [174]. Nevertheless, the most pressing issue, that we are focusing on within this thesis, is the data management for such Big Data applications. In this context, achieving high-performance on the one hand and diversifying the services on the other hand is critical, in order to address the complex challenges brought by the data deluge [27, 64, 184].

A particular complex and difficult aspect of the data management for Big Data applications is handling data across wide areas and/or across the geographically distributed cloud data centers. Nowadays, an increasing number of processing scenarios and applications require such computing, as discussed also in Chapter 2. The main reasons why applications need to geographically distribute the computation on the cloud are the following.

- The *size of the data* can be so big that data has to be stored across multiple data centers. We reiterate the example, described in Chapter 2, of the Atlas CERN experiment [13] which generates 40 PB of data per year.
- The *data sources* can be physically distributed across wide geographical locations. This is the case for the emerging scientific virtual observatories (e.g., ocean [179] or astrological [137] observatories).
- The *nature of the analysis*, which requires aggregating streams of data from remote application instances for an increasing number of services. Large-scale services like search engines or online office tools operate on tens of data-centers around the globe. Maintaining, monitoring, asserting the QoS of the system or global data mining queries all require inter-site (stream) processing.

Our objective is to provide data management solutions which enable *high-performance* processing at *large-scale* across the geographically-distributed cloud infrastructures. In this landscape, building functional and performance-efficient data management solutions, first requires to identify the main requirements that scientific Big Data applications bring when migrated to the clouds.

4.2 Challenges and Issues

Never before the distributed storage and data management systems had to cope at such extent with challenges such as the data volumes and processing throughput that are associated to the emergence of Big Data. The cloud storage services, now in operation, are relatively new and still evolving. So far, they have mainly focused on the needs of business applications, targeting to provide basic functionality in a reliable and secure way. Supporting data-intensive applications in clouds, at large-scale, raises the need to address the following challenges.

Transparency, Automation and Self-Configuration. Localizing, managing, moving and processing large amounts of data is a complex task and cannot be explicitly handled

by users or applications. Therefore, the data management system must transparently handle these aspects, leveraging a uniform view of the data space for applications, regardless of the scale at which data is distributed (i.e., within the deployment, in-site or across sites). Yet, configuring the parameters of the data management system, let alone tuning them for optimal performance, is a difficult task as well [106]. Hence, solutions need to be designed that are capable on the one hand to become aware of their environments and self-configure themselves, and on the other hand to operate without user supervision.

Scalability. Cloud computing is an interesting option for Big Data applications as long as the services can scale with the computation needs. Regarding the storage infrastructure, it requires to be able to leverage a large number of resources efficiently (e.g., virtual disks) and aggregate them elastically and continuously in order to accommodate the growing volume or veracity of the Big Data sets. In the same time, the workload needs to be distributed proportionally with the resources (i.e., *load balance*) in order to avoid performance bottlenecks and idle resources [184]. This is equally valid both for managing the data access requests and for scheduling the computation tasks. To achieve scalability and load balancing in the system, it is important to understand to which extent the traditional techniques, designed for smaller scales (e.g., in the field of distributed shared-memory, cluster scheduling) can be leveraged, extended and adapted to the context of the unprecedented scale brought to reality by cloud infrastructures.

Efficiently assembling the results of parallel executions. One the one hand, effectively dealing with Big Data applications requires to partition the computation to achieve a high degree of parallelism. On the other hand, once the sub-problems are solved, the final results need to be assembled. The performance of this operation impacts the overall execution of the application directly. The critical need for providing efficient solutions is motivated by the fact that the complexity of the operation grows with the parallelism degree. This is even more challenging as constantly pushing the scale boundaries is one of the primary focus of today's research.

Low-latency and high-throughput access to data under heavy concurrency. Big Data processing naturally requires a high degree of parallelism for data analysis (i.e., many compute nodes concurrently access, process and share subsets of the input data). This leads to many application instances accessing the cloud storage system to read the input data, to write the results, to report their state for monitoring and fault tolerance purposes and to write the log files of their computation. Such data accesses strongly impact the overall execution time for many scientific applications. In such scenarios it is important to enhance existing cloud storage systems with means to enable heavy concurrent access to shared data, while avoiding the usually high cost of traditional synchronization mechanisms, in order to provide low-latency and achieve high-throughput access to data.

Fine grain access. Even though many applications use unstructured BLOBs to store their large inputs in a single block, the processing is generally performed in parallel on sub sets of the data. This requires to access small parts of that input data, at a fine grain level. However, large distributed storage systems tend to handle data in large blocks

(e.g., 64 MB) [85, 131], which makes the access to lower granularities rather difficult and inefficient. The goal is to improve the overall data access performance by avoiding useless locking and by dynamically finding and adjusting the right granularity to handle data.

Fault tolerance and data availability. Faults are unavoidable at large-scale in clouds, which are mostly built out of commodity hardware. Therefore, both processing and data management systems need to be designed to account for failures, as they are the norm rather than the exception [85, 173]. Regarding the processing phase, the techniques vary from speculative executions of tasks[82], to using redundancy mechanisms to ensure that the jobs are executed at least once, as with Azure Queues, or adopting checkpointing techniques for cloud processing [133]. In terms of data storage, the usual approach to deal with this issue is data replication [35]. Understanding which options to combine is fundamental to ensure successful execution of large, Big Data scenarios.

Stream processing. Stream processing refers to the management and analysis in real-time of continuous sequences of relatively small data items [68], otherwise referred to as events. Stream processing is becoming one of the most significant classes in the world of Big Data, considering the vast amounts of data which is collected in real-time, at increasing rates in various scenarios. Handling such data is a distinct challenge than managing static or stored data due to the continuous nature of the stream and the typically small sizes of the events forming it. In fact, an extended survey over thousands of commercial jobs and millions of machine hours of computation, presented in [87], has revealed that the stream process performance is highly sensitive to the management and transfer of events, and their latencies. Therefore, there is a growing need for a high-performance system for event streaming on clouds [22, 39, 68].

Data sharing. For many application scenarios, particularly the data-intensive ones, data sharing is the most difficult and compelling of all the enumerated challenges [24]. Whether the Big Data processing is performed in-site or across multiple data centers, the application input needs to be shared across the compute instances, which in turn need to share their (partial) results among each other, with other applications within a workflow or make them available to users. A great deal of effort was invested to propose alternative solutions that would limit the sharing of data across nodes, by leveraging data locality and by moving the computation instead [82, 163]. However, despite these efforts, it was shown that for many applications, the input data can account only for 20 % of the total IO, the rest corresponding to data transmitted during the computation across the instances [87]. This emphasizes that the processing services, be they distributed across multiple nodes, exchange large amounts of data. Additionally, the performance of the computation is governed by the efficiency of the data management. Therefore, providing high-performance data management services is a key milestone for supporting Big Data processing on the clouds.

This list is clearly not exhaustive. Considering the dynamic ecosystem of Big Data applications, new issues are expected to appear both from business as well as from the data science scenarios. Even so, addressing these issues by providing new data management solutions is essential in order to cope with the Data Deluge and to prepare for the next challenges of Big Data. These challenges that we identified serve as a roadmap for the approaches that

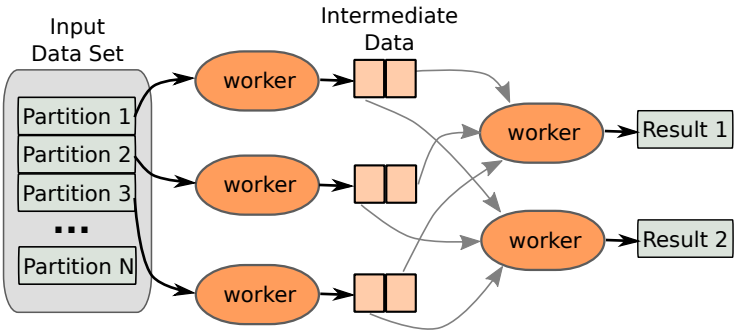


Figure 4.1: The classical MapReduce scheme, as introduced in [46]

are presented in this thesis. However, a solution to an issue is as good as its usability potential. Therefore, it is important to discuss also the processing paradigm(s) that can apply and use the solutions to these issues. Hence, we discuss next the processing models that gather, and, to some extent, shape many of the recent efforts made to address these challenges.

4.3 Big Data Processing Models: MapReduce and Beyond

With the emergence of Big Data, for many applications the computation scenario becomes straightforward: apply a search strategy for a particular property (i.e., the Value) on the input data set. However, as the data sets are typically large (we recall the Volume challenge of Big Data discussed in Chapter 2), the computation needs to be partitioned and distributed across as many resources as possible to obtain acceptable execution times. The classical processing models have to evolve. On the one hand they should facilitate the execution of many parallel tasks by elastically scaling the computation. On the other hand, they should enable the execution of these tasks over a virtualized infrastructure of commodity hardware, such as the cloud. Consequently, the processing models for Big Data can be conceptually abstracted by the orchestrating scheme they propose for the parallel tasks.

MapReduce is a data processing paradigm, which became the "de-facto" model for executing Big Data applications on large infrastructures. The MapReduce model, inspired from functional languages, is shown in Figure 4.1 as it was initially proposed by Google [46]. It consists of 2 functions: *map* and *reduce*, executed in 2 successive phases by worker processes. The *map* function is applied, in parallel, on all the partitions of the input data. Example of map functionality are: counting items occurrences, ranking records, searching for patterns etc. The *reduce* function has the role of assembling the intermediate, partial results of the mappers into the final output. The assignment of the intermediate results by the mappers to the reducers is done based on a hashing function. Typical examples of reduce operations are: sum, sort, filter, merge, etc. The large success and popularity gained by this model is due to the fact that it effectively fulfills the 2 requirements previously mentioned for Big Data processing: easy scalability and potential to be executed on virtualized commodity hardware. Users only need to provide the 2 functions, which are then scaled with the number of resources and applied on the partitioned input set. Additionally, since the computation is performed on independent sub sets, the model has a natural tolerance to failures. In fact,

it enables to simply re-execute a failed task on a different machine. In this way, the model can accommodate potential failures of the commodity hardware, without any effort or infrastructure knowledge, thus allowing it to run on virtualized cloud environments.

MapReduce extensions were developed due to the success of the MapReduce model, and with the goal of extending its scheme to accommodate new types of processing, while preserving its initial properties. Among these, the most representative extensions are the *iterative MapReduce* processing and the *pipeline of MapReduce* processes. Iterative MapReduce applies the pair of map and reduce functions in several iterations on the data (i.e., the output data from reducers is feed back to mappers). The termination condition is usually determined either by the number of rounds elapsed or by evaluating the differences between the outputs from successive iterations and stopping when these become smaller than a threshold. When data cannot be processed in successive iterations with the same functions, the MapReduce pipelines become an interesting option. The idea is to orchestrate several independent MapReduce processes, each with it's own map and reduce operations, in a processing flow [128]. Unlike for the iterative declination, creating pipelines requires a meticulous instrumentation and more overhead, thus tempering the initial goals of MapReduce. Others extensions, which do not change the data flow model, propose optimization by moving the computation in-memory, processing data incrementally or in batches from a live data input stream [84, 155].

Beyond MapReduce: Workflows are migrated to the clouds because, even with the extensions to the initial MapReduce model, many scientific applications cannot fit this computation paradigm. Nevertheless, the tendency is to mix the expressivity provided by workflows with the simplicity of MapReduce. The goal is not only to allow users to create more diversified dependencies between tasks, but also to simplify the description of the interdependencies and the specifications of the data flow. At the same time, efficiently handling generic (and complex) orchestrations of data flows on a virtualized environment raises an important challenge for the data management systems. In fact this becomes the key point for the overall efficiency of the system. Hence, providing efficient data services capable to interact with the processing engines and to enable high-performance data exchanges between the compute nodes is a key milestone for Big Data processing on clouds.

4.4 Discussion

Porting data intensive applications to the clouds brings forward many issues in exploiting the benefits of current and upcoming cloud infrastructures. In this landscape, building a functional infrastructure for the requirements of Big Data applications is critical and is still a challenge. We investigated and identified several hot challenges related to Big Data management on clouds and discussed the main processing models that can leverage solutions to these issues for applications. We use this analysis as a motivation and design guideline for the data management approaches that we propose in this thesis, in order to enable *high-performance* processing at *large-scale* across the geographically-distributed cloud infrastructures. The next chapter introduces the current state-of-the-art solutions in their domains and discusses their limitations in meeting these challenges.

Chapter 5

State-of-the-Art: Big Data Management Systems for Clouds

Contents

5.1 Data Storage 33
5.2 Data Processing 36
5.3 Transfer 41
5.4 Discussion 43

With the migration of Big Data applications towards large-scale infrastructures such as the clouds, several solutions have emerged. In this chapter we discuss the main approaches proposed, both from data management and processing points of view. The goal is to depict their strengths and identify the issues that still remain open for Big Data management on clouds, which is the primary focus of this thesis.

5.1 Data Storage

Being able to store efficiently the large volumes of data produced today is one of the primary challenges raised by Big Data. To this end, several solutions were proposed for large-scale infrastructures. Depending on the data format they employ (i.e., whether they use or not a structure), the storage solutions can be divided into two categories, which we discuss next.

5.1.1 Unstructured Data: Object-Based Storage Systems

Storing data in large binary BLOBs is the main approach for dealing with Big Data. Cloud providers offer such object storage solutions (e.g., *Amazon S3* [9], *Azure Blobs* [125], *Google*

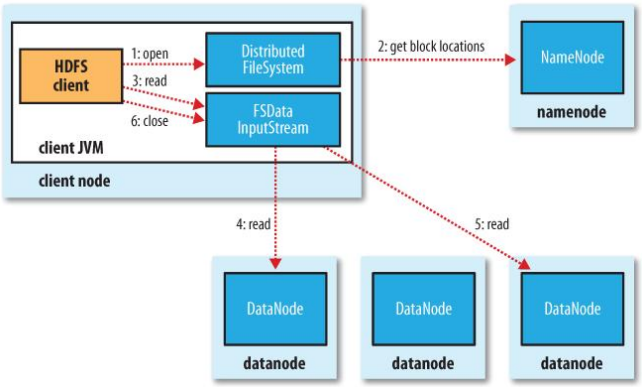


Figure 5.1: HDFS Architecture [177]

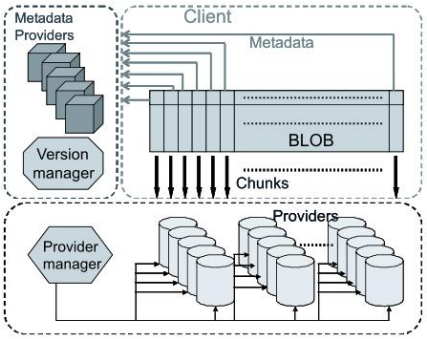


Figure 5.2: BlobSeer Architecture [131]

Cloud Storage [71]) as the primary service for handling data, as discussed in Chapter 3. They are optimized for high availability and data durability, under the assumption that data are frequently read and only seldom updated. Therefore, achieving high throughput or enabling optimizations for Big Data applications (e.g., by exposing the data layout) are auxiliary goals. Moreover, there is a clear separation within the data center, between these systems and the computation infrastructure.

5.1.2 Structured Data: Distributed File Systems

Having the goal to move the computation closer to the data, several distributed storage solutions were developed. These share the common principle with the cloud storage, of storing data distributed across the system entities, in large blocks. However, they mainly target the computation infrastructure rather than a specialized data storage hardware [35].Such systems are usually deployed over the node file system. Nevertheless, they are not generally available for default users on clouds. Next, we discuss three representatives solutions, as follows.

HDFS (Hadoop Distributed File System) [85] was developed to serve as the storage backend for MapReduce compute platforms. It has a master–workers architecture, as depicted in Figure 5.1. The centralized control component, called NameNode, manages all metadata within the system, and additionally splits and disseminates the data for storage. The nodes holding the data are called DataNodes. Data are typically split and stored in chunks (64 MB size is commonly used during the MapReduce process). Regarding fault tolerance, chunks are replicated across several DataNodes (by default, 3). As optimizations, HDFS exposes the data layout, offers data buffering capabilities and advanced provisioning, allowing the compute frameworks to schedule the computation accordingly. However, HDFS cannot sustain a high throughput for concurrent reads [132]. Also, concurrent writes or appends are not possible.

BSFS/BlobSeer [132] is a concurrency-optimized distributed storage system for large binary objects, which can be used via a file system interface. It consists of a set of distributed entities that enables scalable aggregation of the storage space from the partici-

pating nodes with minimal overhead. Data striping and replication is performed transparently for applications, while a version-oriented metadata scheme enables lock-free access to data, and thereby favors scalability under heavy concurrency. The architecture of BlobSeer is shown in Figure 5.2. Data providers physically store the blocks corresponding to the data updates. New providers may dynamically join and leave the system. The provider manager keeps information about the available storage space and schedules the placement of newly generated blocks, according to a load balancing strategy. Metadata providers store the information that allows to identify the blocks that make up a version of the data. The version manager is in charge of assigning version numbers in such a way that serialization and atomicity are guaranteed. The system is capable of delivering high throughput performance, but requires meticulous configuration, which is not always straightforward for applications designers.

GFarm [136] is a distributed file system, enabling data sharing for data-intensive computing. The system was initially designed for Grid infrastructures, and therefore stands as a good example for the storage approaches that were refocused and migrated into clouds. However, integrating it with a cloud middleware requires to run it on the host OS and specifically modify and extend the cloud middleware. Therefore, such solutions are feasible only in the context of private infrastructures, which are virtualized with open-source cloud toolkits.

Such storage solutions have proven to be effective in some application scenarios. However, they are not always ready to be used out-of-the-box for scientific Big Data processing on clouds, particularly in the case of public ones. Moreover, obtaining optimal performance usually requires information about the environment, which by default is not explicitly accessible in the user virtualized space.

5.1.3 Structured Data: Key-Value Stores

The demand for storing large volumes of data in a structured fashion cannot be met by the traditional relational databases, as mentioned in Chapter 3. Consequently, systems, generically called NoSQL, were proposed to enable storing large amount of data in the form of key-value pairs, and therefore are called *key-value stores*. Some of these systems enable the organization of the items in rows and columns, providing partial indexes with respect to this scheme. Due to their efficiency in storing and managing massive data collections, obtained from the lack of the full ACID (Atomicity, Consistency, Isolation, Durability) guarantees, such solutions were built by companies, which own and manage large-web services.

Azure Tables [35] is the structured key-value storage offered by Microsoft as a cloud service. A Table contains entities (similar to rows in classical relational databases), which store multiple properties. Each entity can contain a different number of properties or different types of properties. At least 3 properties are always present: RowKey (the unique identifier), Timestamp (used as a versioning mechanism) and the PartitionKey (used to physically group records together). The latter is used by the system to balance the load across multiple storage nodes and to optimize consecutive accesses for performance.

Cassandra [113] is a distributed storage for structure data, built by Facebook. The system was designed for high scalability on commodity hardware withing or across data centers. It resembles a relational database in several aspects, storing data in tables with row index capabilities, but with no further support for a full relational model. Multiple replicas are created and distributed for fault tolerance across multiple physical nodes. Based on them, the system implements customizable consistency levels (i.e., number of replicas considered when performing an operation).

BigTable [40] is the distributed NoSQL storage introduced by Google, designed to scale to Petabytes of data and thousands of machines. As for Cassandra, it does not support a full relational model, offering instead dynamic control over layout and format of tuples and advanced locality management. Similar with Azure Tables, data is organized based on 3 properties: rows, columns and timestamps.

PNUTS [44] is the geographically distributed storage provided by Yahoo!. It provides a simple and effective model, with data organized in hashed or ordered tables. Tables contain records, augmented with attributes, as the basic unit for the data.The tables are partitioned into smaller data structures which are stored across servers from different regions. The concept of regions is similar with the abstraction of fault domain of the Azure cloud, and refers to the partition of the overall infrastructure. However, the resulting regions can also be geographically distributed.

Dynamo [47] is the highly available internal storage for the Amazon web services. The system is highly optimized for writes and for low latencies. It is organized as a distributed hash table across a ring of virtual nodes, which have a many-to-one mapping with the physical nodes. The data model relies on simple key-value pairs, which are asynchronously replicated and eventually reconciled based on versions. The client applications are offered a flexible control for the consistency levels of accessing the replicas, and thereby on the cost effectiveness.

These NoSQL systems are designed for massive scalability and therefore are good candidates for managing the Volume challenge of Big Data. To a great extent, these are among the most advanced state-of-the-art systems in terms of volumes stored, reaching the order of Petabytes. Due to their high business value, most of them are closed, proprietary systems, such as Dynamo or BigTable. Typically optimizing for a certain consistency level or availability guarantee for the data, they do not always deliver the highest performance. Moreover, properties such as data locality are not considered or not applicable, specially considering that their initial goal is to scale over large (geographically) distributed infrastructures.

5.2 Data Processing

Data processing is another important aspect that needs to be considered in the context of Big Data, as understanding its characteristics helps to design the proper solutions. However, with the diversification of the compute scenarios, different processing approaches are, and need to be, ported on the cloud or on large infrastructures, in order to meet the application challenges. The most popular of them is the MapReduce processing. Alongside with systems that implement this paradigm, we discuss solutions for general-purpose or real-time

processing, in order to extend the applicability areas of the data management solutions considered in this work.

5.2.1 MapReduce

We recall that the MapReduce processing paradigm, introduce by Google [46], is presented in Chapter 4. In what follows, we present the state-of-the-art solutions, which adopt this computing model.

From Hadoop to YARN and Spark. The most notorious and used implementation of the Goggle MapReduce [46] is the *Apache Hadoop* [82]. This distributed framework runs on top of HDFS storage, collocating computation and data. It proposes a master-worker architecture for scaling the processing of data sets across clusters, while providing strong fault tolerance guarantees (speculative execution of tasks, heartbeats mechanisms, re-execution of failed tasks, etc.). The two components of the system are the Job Tracker, which is in charge of scheduling and managing the execution of the MapReduce tasks and the TaskTrackers, which run on each compute node and are in charge of executing the computation tasks. To increase efficiency, map and reduce tasks are scheduled according to the locality of data, with the goal of reducing data movements across nodes and racks. As in the original model, the map and reduce phases are separated, avoiding deadlocks though a synchronization barrier between the two stages.

Because of its success, the framework was adopted and used in diverse scenarios beyond its initial designed goals. For example, high-level frameworks such as Pig [145] or Hive [94], or full workflow descriptive languages such as Kepler [175], orchestrate complex and heterogeneous MapReduce execution plans. This extensive use, as well as a vast number of research studies, revealed several limitations of the architecture such as tight coupling of the processing model with the infrastructure or scalability and performance bottlenecks due to the centralized scheduling. To address these limitations, a new service-oriented architecture, called *YARN* [170], was proposed, which is depicted in Figure 5.3. The initial multiple roles of the JobTracker are split between multiple Resource Managers, which administrate entire clusters, and the Application Master, which schedules and coordinates the execution of the work per application. The compute infrastructure is divided into discrete quotas called containers, which are managed by entities called Node Managers. In this way, YARN provides better resource utilization, higher fault tolerance and greater scalability.

An alternative MapReduce system implementation, which gain popularity recently, is Spark [155]. It accelerates the computation building on the observation that clusters can support the computation in-memory. Therefore, it proposes a MapReduce architecture which handles and processes data entirely in-memory, being thus limited within the available memory of the cluster. Moreover, the framework proposes extensions also to the MapReduce API enabling to launch computation as SQL queries [182].

MapReduce on clouds. Due to the impact of the MapReduce on Big Data processing and the popularity of Hadoop, the model was also ported to the clouds. AzureMapReduce [77] is one of the first attempts to address the need for a distributed programming framework in Azure. It is a MapReduce runtime designed on top of the Microsoft Azure PaaS cloud

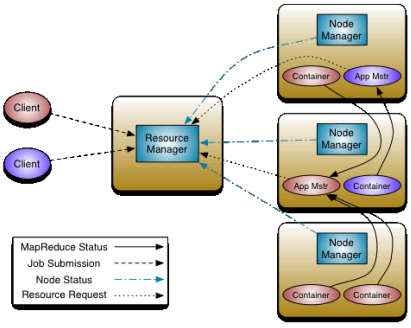


Figure 5.3: Apache Hadoop NextGen MapReduce (YARN) [82]

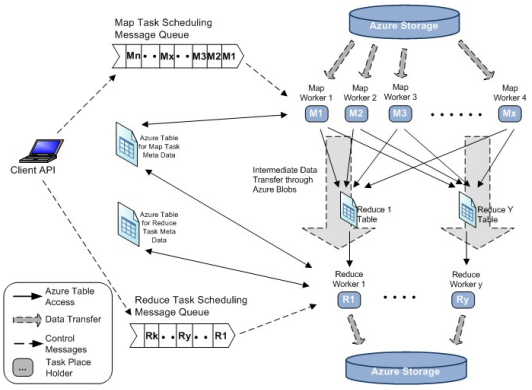


Figure 5.4: AzureMapReduce architecture [77]

abstractions, which uses the cloud storage services for data and task management (e.g., Azure Queues for scheduling, Azure Tables for metadata administration). These choices provide great fault tolerance, but reduce the performance of the service. Later than this work, Microsoft proposed also an own MapReduce service based on Hadoop, called HDInsight [86], following the model of Amazon which offers its Elastic MapReduce service [8]. Both these cloud-provided services externalize the processing from users. In this way, the task of running single MapReduce computations is simplified, but the services limit potential optimizations based on the scenario semantics as well, as integrating the processing into larger computation pipelines.

Extensions of MapReduce. The wide adoption of the model served as a motivation for several efforts to propose extensions that would enable to process a larger class of applications. The main focus was on the iterative ones. As Hadoop does not support such iterative processing by design, HaLoop [32] was built on top of it with this purpose. It exposes a new programming model and relies on a loop-aware task scheduler and on loop-invariant data caching. Besides HaLoop, other solutions accelerate iterative algorithms by maintaining iteration state in memory. Twister [53] employs a light weight MapReduce runtime system and uses publish/subscribe messaging-based communication instead of a distributed file system. Mappers and reducers are built as long-running processes with distributed memory caches in order to avoid repeated data loading from disks. The solution was also ported to Azure, by extending the aforementioned AzureMapReduce framework [78]. iMapReduce [185] extracts common features of iterative algorithms and provides support for them. In particular, it relies on persistent tasks and persistent socket connections between tasks, it eliminates shuffling static data among tasks, and supports the asynchronous execution of iterations when possible. All of these frameworks target applications with iterations across MapReduce jobs and require additional components and programming efforts to aggregate their output automatically. Moreover, the close loop architecture is in most cases sensitive to failures. Additionally, potential in-node optimizations of the processing, such as caching data between iterations, might not scale in commodity machine clusters, where each node has limited memory and resources.

5.2.2 Workflows

A large number of workflow engines were developed due to their impact and efficiency in solving business processes. Enumerating all these solutions or discussing their general features is beyond the purpose of this work. Our focus is to identify the main characteristics that such engines share, when ported on the clouds, as traditionally, they run on private or grid infrastructures. Therefore, we discuss three such engines which were either ported or constructed to run on cloud infrastructures.

Pegasus [48] is a traditional (i.e., not designed for clouds) workflow engine, which facilitates the representation of data-compute flows in an abstract manner and executes them on distributed computational resources. The framework optimizes the mapping of the sub-tasks, in which the workflow is divided, with the resources, with the goal of minimizing the data movements and improving the execution times. However, when migrated to the cloud, its performance is highly sensitive to the data management solution used, which, to a large extent, determines the scheduling optimizations techniques that can be applied [2]. Providing cloud services that can leverage infrastructure topology and data locality information are key points that need to be addressed to increase the feasibility and efficiency of running such workflow engines on clouds.

e-Science Central [91] was designed as a workflow platform accessible via web browsers, following the "'as a Service" cloud model. It enables collaborative work, as well as data and computation sharing. Workflow execution is performed as a sequence of service invocations. The corresponding execution plan is determined and processed as independent tasks across the distributed compute resources. The orchestration of the execution is performed by a centralized coordinator which limits the scalability of the computation and its performance. A second factor impacting the performance is the use of the cloud-provided storage service, which recent efforts are trying to replace with other solutions such as in-nodes HDFS [92].

Generic Worker [153] stands as an example of a workflow engine specifically designed for running scientific applications on clouds. It was implemented as an open-source, batch-based processing platform by Microsoft Research ATL Europe [66]. The system can seamlessly be extended by the use of plugins and can be migrated to any cloud, private or public. This flexibility was enabled in order for researchers to apply a hybrid approach: use a local infrastructure when available, and scale-out to the cloud when necessary. By default, Generic Worker relies on the cloud storage to perform all data management operations such as tasks description or data uploading and sharing. On the one hand, such a design choice supports the extensibility of the processing across infrastructures, but on the other hand it provides low I/O performance for data transfers between tasks, compute nodes and infrastructures. This indicates that such elastic Big Data processing solutions, potentially operating across geographically distributed infrastructures, require dedicated cloud services for wide-area data management and transfers, considering that the cloud-provided storage service offers such functionality only circumstantially.

5.2.3 Complex Event Processing Platforms

The previous discussed processing methods are mainly designed for analyzing static collections of data. However, real-time stream processing becomes an important aspect of Big Data, with massive quantities of information being produced and requiring on-the-fly processing (e.g., see the Velocity challenge, discussed in Chapter 2). Such analysis is handled by specialized platforms, called complex event processing (CEP). Due to the complexity of partitioning a query (i.e., the description of the computation task), most of such systems were designed with centralized architectures and did not consider the cloud infrastructures. Hence, a hot open-issue in the area today is how to port and scale these engines to large cloud infrastructures.

Middleware solutions like System S from IBM [65] were proposed for single cluster processing, with processing elements connected via typed streams. The ElasticStream system [101] migrates this solution to Amazon EC2 taking into consideration cloud-specific issues like SLA, VMs management and the economic aspects of performance. Similarly, Microsoft provides StreamInsight [127], which is available for installation into Azure cloud machines. Other works in the area of large-scale streaming, like Sphere [75], propose a GPU-like processing on top of a high-performance infrastructure, which restricts the general applicability to clouds. Systems like Aurora and Medusa [1] consider exploiting the geographically distributed nature of the stream data sources. However, the systems have a series of limitations despite their strong requirements from the underlying infrastructure (e.g., naming schema, message routing): Aurora runs on a single node and Medusa has a single administrator entity. The main issue with this series of systems is that they were not designed for cloud and/or large-scale processing, and consequently they cannot effectively leverage the underlying platform for high-performance computing. Bellow are three other systems which propose alternative directions.

D-Streams [182] provides tools for scalable stream processing across clusters, building on the idea of handling data in small batches, which can be processed using MapReduce. This idea also discussed in [122]. In this way, the scalability problems of stream processing are addressed by relying on the elasticity of the MapReduce processing engines. Data acquisition is event-driven: the system simply collects the events from the source and batches them. Next, a MapReduce process is launched to execute the query on the batched events. Although this approach solves the scalability issue, it provides a high latency-to-answer, being more adequate for applications that tolerate to obtain the answers with some delay, i.e., in near real-time.

Stormy [120] implements concepts from peer-to-peer stream processing in the context of clouds. The system is an elastic distributed-processing service that enables running a high number of queries on continuous streams of events. The queries are replicated and applied on all the replicas of the events created across the peer nodes. Stormy delegates the staging-in of the events to the data source which is expected to push the events in the system. Moreover, it handles the stream as individual events both in the acquisition and the replication phases, which may drastically reduce the overall performance.

Storm [165] is an open-source real-time processing engine developed by Twitter for large-scale processing. The system builds on several existing solutions to distribute the event

processing while guaranteeing fault-tolerance. Resilience is accounted for in the design alongside with efficiency, Storm operating in-memory with one of the "at least once" or "at most once" processing semantic guarantees. Data is handled at the level of tuples, over TCP, by 2 threads per worker: the sender and the receiver. Tuples are managed in queues, being gathered (i.e., from the local tasks executed on the node or from different nodes), independently inspected and finally delivered towards the destination. In addition to handling the stream in memory, Storm uses the local disks of nodes to store the processing topology (the computation and the description of the data flow), states and the user code/query.

All these solutions focus on processing queries and scalability with no specific improvements nor solutions for the event streaming itself. This raises a major difficulty for addressing the tough challenges related to the distributed nature of the analysis or of the stream source locations.

5.3 Transfer

So far, we discussed the main processing approaches and their storage options. The primary feature identified is *scalability*, which translates into managing data in a highly distributed fashion. In this context, the transfer of data becomes particularly important whether data is replicated to increase its resilience or shared between tasks or computing phases for processing purposes.Hence, we discuss next the state-of-the-art solutions for such data movements. We divide them according to the state of the data, which can be either static or in motion, as part of a continuous flow (i.e., stream).

5.3.1 Static Data

The handiest option for sharing or transferring data, even across cloud data centers, is to rely on the cloud storage services. This approach allows to transfer data between arbitrary endpoints and it is adopted by several systems in order to manage data movements over wide-area networks [111, 129]. However, this functionality arises only as a "side effect", and therefore achieving high transfer throughput or any other potential optimizations, such as differentiated QoS or cost effectiveness, are not always viable. Trying to fill this utility gap, several alternatives have emerged.

NetSticher [116] was designed for bulk transfers of data between data centers, with the purpose of replicating for geo-redundancy. The authors exploit the day/night pattern peaks of usage of a data-center in order to leverage the unutilized bandwidth periods. Though NetSticher is useful for backup and checkpoint operations, required to administrate a data center, it does not work for real-time systems nor for online applications which need to exchange data between their running instances.

GlobusOnline [58] provides file transfers over WAN, targeting primary data sharing between different infrastructures (e.g., grids, institution infrastructures, community clouds), mainly within scientific labs. The framework is designed as a SaaS with the focal point on the users, which are offered intuitive web 2.0 interfaces with support for

automatic fault recovery. GlobusOnline runs based on the GridFTP [3] tool (initially developed for grids), performing file transfers only between such GridFTP instances. During the transfers, it remains unaware of the environment and therefore its transfer optimizations are mostly done statically. Several extensions brought to GridFTP allow users to enhance transfer performance by tuning some key parameters: threading in [118] or overlays in [109]. Still, these extensions only focus on optimizing some specific constraints, which leaves the burden of applying the most appropriate settings to scientists, which are typically unexperienced users.

StorkCloud [112] offers also a SaaS approach for data management over wide-area networks. It integrates multi-protocol transfers in order to optimize the end-to-end throughput based on a set of parameters and policies. It adapts the parallel transfers based on the cluster link capacity, disk rate and CPU capacity, using the algorithm proposed in [181]. The communication between StorkCloud components is done using textual data representation, which can artificially increase the traffic for large transfers of unstructured data. The system is one of the most representative efforts for improving data management. It is designed for general purpose, targeting the communication between any devices. Ergo, specific optimizations for the Big Data scenarios running on clouds are not considered, such as the topology of the applications in the data center(s) or of the inter-connecting links between compute nodes.

Multi-path TCP [148] is a standard designed to address the challenge of moving large volumes of data by enabling parallel data transfers. The idea is to improve the communication by employing multiple independent routes to simultaneously transfer disjoint chunks of a file to its destination. The approach can incur additional costs such as higher per-packet latency due to timeouts under heavy loads and larger receiving buffers. Nevertheless, this remains an interesting solution for tackling Big Data processing. However, it is designed to operate at the lower levels of the communication stack, thus ensuring congestion control, robustness, fairness and packet handling. Consequently, the solution is not available for cloud users until it will be adopted by cloud vendors and Tier 1 ISPs.

These efforts show that the need for efficient tools for managing data is well understood in the community and stands as an important issue. Nevertheless, many aspects remain unaddressed, particularly finding solutions that would provide high-performance for transfers between the running instances of applications on the cloud.

5.3.2 Real-Time Data

Handling data in real-time, for large-scale Big Data processing, calls for different solutions than the ones previously discussed for general data transfers. These need to be specifically designed for continuous streams of events, together with the CEP engines and considering the target computation environments. However, despite the growth in volumes and importance of stream data, and the efforts to migrate such processing to the clouds, most of the existing stream processing engines only focus on event processing and provide little or no support for efficient event transfers between even source and processing engine. In fact, this functionality tends to be delegated to the event source [31, 182]. As a result, the typical way to transfer events is individually (i.e., *event by event*), as they are produced by the

data source. This is highly inefficient, especially in geographically distributed scenarios, due to the incurred latencies and overheads at various levels (e.g., application, technology encoding tools, virtualization, network). Therefore, currently there is a lack of support and focus for optimizing the event streaming and real-time transfers on clouds. This is explained somehow by the fact that the topic of processing real-time data on clouds is new, and thus not all aspects have been addressed yet.

In other areas, where real-time data management has been around for some time, things are more advanced. For example, the systems providing *peer-to-peer streaming* can be divided in two categories based on how peers, which forward the events, organize themselves in an overlay network [159]: some use DHT overlays [79, 157] and others group the subscribers in interest groups based on event semantics [160, 161]. While the performance of the former is highly sensitive to the organization of the peers, the latter can improve the performance by sharing common events within the interest group. Further optimizations can be achieved by discovering the network topology which is then matched to the event traffic between the subscribers [159]. This aims to improve the network usage by identifying whether independent overlay paths correspond to common physical paths and by allowing deduplication of events. However, in the context of clouds, it is not possible to consider information about physical paths, as knowledge and interactions are limited to the virtual space. Moreover, these techniques do not consider staging-in data from the source to the CEP engines, but rather disseminating information among subscribers.

Similarly, in the area of *video streaming* there were significant efforts to improve the end-to-end user experience by considering mostly video specific optimizations: packet-level correction codes, recovery packets, differentiated redundancy and correction codes based on the frame type [96]. Additionally, such solutions integrate low-level infrastructure optimizations based on network type, cluster knowledge or cross-layer architectures [114, 123]. Nevertheless, none of these techniques directly apply to the Big Data real-time processing on clouds, as they strongly depend on the video format, and thus are not applicable for generic cloud processing.

The straightforward conclusion is that there is a critical need for high-performance transfer solutions for real-time data. This conclusion is also confirmed by other studies [22]. What makes it critical is that the value of the real-time analysis itself (i.e., see the Veracity challenge of Big Data discussed in Chapter 2) is determined by the performance of transferring the data to the processing engine. Sharaf et al [152] emphasize this point stating that *data freshness* improves the Quality of Service of a stream management system.

5.4 Discussion

Supporting Big Data processing and Data Science strongly relies on the ability to store and share massive amounts of data in an efficient way. Cloud infrastructures are a key element in this process and must therefore address the challenges posed by applications whose requirements are becoming more and more demanding (e.g., in terms of performance and scalability at global level), as discussed in the previous chapters. In this section we presented an overview of the current status and solutions for data management on cloud infrastructures. The survey has revealed that the processing frameworks strongly rely, in their pursuit for scaling, on the data management backends. It also showed that in terms of storage, the cloud

landscape offers several solutions to support data processing, mostly focusing on providing guarantees such as persistence and fault tolerance. However, aspects like performance or diversification of the data-oriented services tend to be overlooked. Regarding the support for managing data movements for the applications running on the clouds, things are worse. There is a clear need for developing solutions that would enable efficient data sharing across compute instances, in order to sustain the scalability required to accommodate the Big Data processing. In this thesis, we address these open challenges regarding data management, by proposing a set of high-performance data services for storing, sharing and transferring data on clouds, both for single and multiple site processing.

Part II

High-Performance Big Data Management on a Single Data Center

Chapter 6

MapReduce for Bio-Informatics: The A-Brain Case Study Application

Contents

6.1 Joining Genetic and Neuro-imaging Analysis 47
6.2 Towards a MapReduce Architectural Solution 51
6.3 Discussion 52

This chapter develops the contributions published in the following paper:

- *A-Brain: Using the Cloud to Understand the Impact of Genetic Variability on the Brain.* Radu Tudoran, Alexandru Costan, Benoit Da Mota, Gabriel Antoniu and Bertrand Thirion. Microsoft Cloud Futures Workshop 2012, Berkeley, CA, US

6.1 Joining Genetic and Neuro-imaging Analysis

An increasing number of scientific Big Data applications are being ported to the cloud to enable or accelerate the scientific discovery process. To better evaluate the needs and identify the requirements that such applications have from large-scale cloud infrastructures, we selected a representative application from the bio-informatics domain, called A-Brain. Next, we present this application and the challenges that it brings for clouds, which we address in the following chapters.

6.1.1 Initial Motivation

Joint *genetic* and *neuroimaging* data analysis on large cohorts of subjects is a new approach used to assess and understand the variability that exists between individuals. This approach has remained poorly understood and brings forward very significant challenges, as progress in this field can open pioneering directions in biology and medicine. The goal is to enable joint analysis between the domains, as well as understanding and explaining aspects from one area based on the other, similar with what is possible today between the clinical behavior on the one hand and either genetics or neuroimaging on the other. However, both neuroimaging- and genetic-domain observations involve a huge amount of variables (i.e., in the order of millions). Performing statistically rigorous analysis on such Big Data has high scientific **Value**, but raises a computational challenge that cannot be addressed with conventional computational techniques.

Several brain diseases have a genetic origin or their occurrence and severity is related to genetic factors. Currently, large-scale studies assess the relationships between diseases and genes, typically involving several hundreds patients per study. Thus, genetics plays an important role in understanding and predicting responses to treatment for brain diseases like autism, Huntington's disease and many others. However, it remains poorly understood. Brain images are now used to explain, model and quantify various characteristics of the brain. Since they contain useful markers that relate genetics to clinical behavior and diseases, they are used as an intermediate between the two.

Imaging genetic studies linking functional magnetic resolution imaging (fMRI) data and Single Nucleotide Polyphormisms (SNPs) data are facing the challenges of multiple comparisons. In the genome dimension, genotyping DNA chips allows to record several hundred thousands values per subject, while in the imaging dimension an fMRI volume may contain from 100 thousands to 1 million voxels (volumetric picture elements), as shown in Figure 6.1. Finding the brain and genome regions that may be involved in this correlation entails a huge number of hypotheses. A correction of the statistical significance of pair-wise relationships is often needed, but this may reduce the sensitivity of statistical procedures that aim at detecting the associations. It is therefore desirable to set up techniques as sensitive as possible to explore where in the brain and where in the genome a significant correlation can be detected. More, the false positive detections need to be eliminated, which requires multiple comparisons. Such gains in sensitivity can also be provided by considering several genetic variables simultaneously.

6.1.2 A-Brain: Application Description

The A-Brain application addresses this computational problem by testing the potential links between brain locations, i.e., MRI voxels, and genes, i.e., SNP. Dealing with such heterogeneous sets of data, is also known as the **Variety** challenge of Big Data applications. The analysis relies on a set of sophisticated mining and learning techniques for inspect the targeted data sets. *Univariate* studies find the SNPs and MRI voxels that are significantly correlated (e.g., the amount of functional activity in a brain region is related to the presence of a minor allele on a gene). With *regression* studies, some sets of SNPs predict a neuroimaging/behavioral trait (e.g., a set of SNPs altogether predict a given brain characteristic), while with *multivariate* studies, an ensemble of genetic traits predict a certain combination

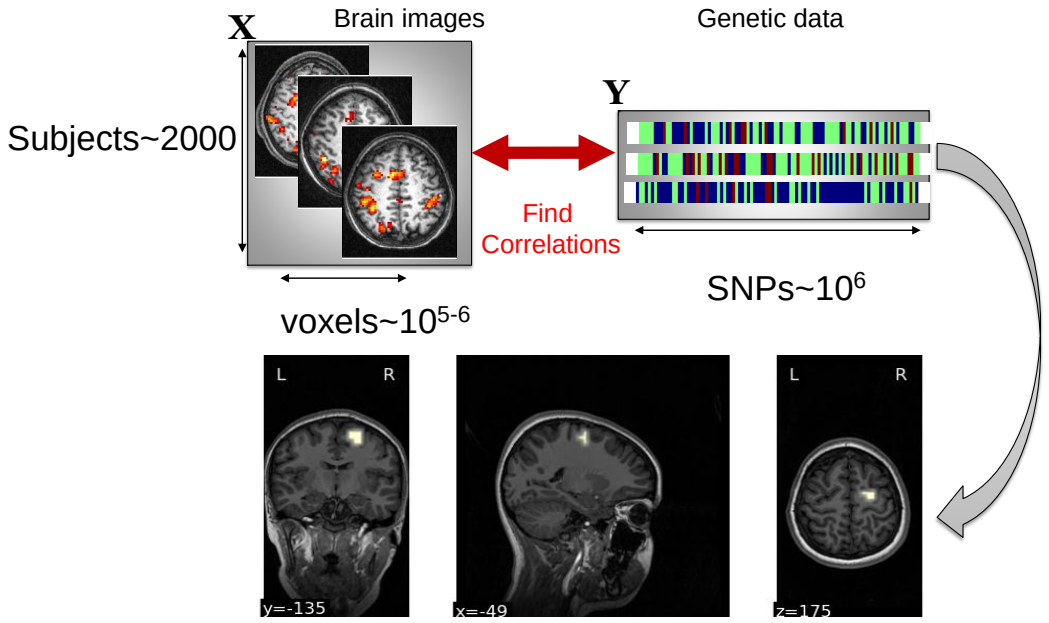


Figure 6.1: The A-Brain dimensional scale for the X (brain images) and Y (genetic data) data sets. The white marks on the brain image illustrate potential associations with the genes.

of neuro-imaging traits.

Let (X, Y) be a joint neuro-imaging data set, as shown in Figure 6.1, i.e., a set X of brain images, that represent the amount of functional activation in response to a certain task or an anatomical feature, such as the density of gray matter, and a set Y of genetic variables (e.g. Single Nucleotid Polymorphisms and/or Copy Number Variants of genetic loci), acquired in the same population of subjects. X is assumed to include n_v variables (e.g., one for each location in the brain image domain), while Y comprises n_g variables, for each of the n_s subjects. This gives two matrices of real values, with typical magnitudes of $n_v \sim 10^6$, $n_g \sim 10^6$. The data set may also comprise a set Z of behavioral and demographic observations, such as psychological tests or age. These variables are not independent.

There are two approaches for evaluating the correlations between the gene and neuro-imaging data. The first approach, the univariate analysis, is less computational demanding than the second one which performs statistics based on machine learning. However, the later is expected to be less sensitive to outliers and therefore more statistically robust. In what follows we present these approaches and their appliances for this application.

- In the first phase of A-Brain, the **univariate analysis** [146] was used to detect the correlations, that is, test the statistical significance of the correlation or the equivalent association measure of all (x, y) pairs for $(x, y) \in X \times Y$. After performing the necessary computations, the correlations between the two sets are obtained, giving a matrix of size $n_v \times n_g$ containing the p-values that represent the statistical significance of the associations. To ensure correctness, that is controlling for false detections, a permutation procedure is used, in which the data of one block is reshuffled ($\sim 10^4$). The p-values obtained in the initial regression analysis are compared afterwards to those obtained from the shuffled sets.

- The alternatively to the univariate analysis is to use a **machine learning technique** in order to increase the precision of the computation and to reduce its sensitivity to outliers, making thus the statistics more robust. Such a computation follows the same principles as before but increases the amount of computation performed. An important finding resulted based on our work and the A-Brain project [45], was to show that such a machine learning approach can compensate, unlike the univariate analysis, for the errors introduced in the acquisition process of human data set. Hence, increasing the computation workload is equivalent with acquiring a data set 100 times larger and thus 100 times more expensive (i.e., 1 billion euros would be needed for such a data set).

6.1.3 Challenges

The available data set used for testing consists of 50,000 voxels and $\sim$0.5 million SNPs for approximately 2000 human subjects. The anatomical brain images are T1-weighted with a spatial resolution $1 \times 1 \times 1$ mm. On the other hand the SNPs were obtained by genotyping, genome-wide using Illumina Quad 610 and 660 chips. Solving this problem leads to the following challenges.

Computation. In order to obtain the results with the expected high degree of confidence, a number of 10^4 permutations are required, resulting in a total of 2.5×10^{14} associations to be computed. Dealing with such complex aspects for the confidentiality of the results constitutes the **Veracity** challenge of this Big Data application. The univariate algorithm, developed by the partner bio-informatic team, performs 1.5×10^6 associations per second. Hence, on a single-core machine the time estimation to run the algorithm is $\sim$5.3 years. Moreover, the alternative algorithm for performing the statistical robust analysis is almost 20 times more slower than the plain univariate method. Thus, in terms of timespan for single core machine, this algorithm would require $\sim$86 years to complete. Luckily, the analysis is embarrassingly parallel, opening the door for substantial improvements.

Data. Following the regression stage which compares the 50 thousand voxels and 0.5 million SNPs for the 2000 subjects, all the intermediate correlation produced must be retained for identifying the most significant p-values. Taking into account that we use matrices of size $n_v \times n_g$ of doubles, the space of intermediate data can reach 1.77 PB. Moreover, the nature of the analysis require the input data set, which is in the order of several of GBs, to be replicated on each node that executes a task. Hence, the size of the input data space is proportional to the degree of parallelism, rapidly reaching the order of TBs. These data dimensions make the analysis challenging from the **Volume** perspective of the Big Data applications.

Environment. The application requires a Python-based scientific environment to be set up, together with scientific libraries such as NumPy, SciPy and H5Py or the Scikit-Learn toolkit. A complex computing environment is also needed for configuring and scheduling the computing jobs automatically, staging in the input data, transferring the data between the nodes and retrieving the results.

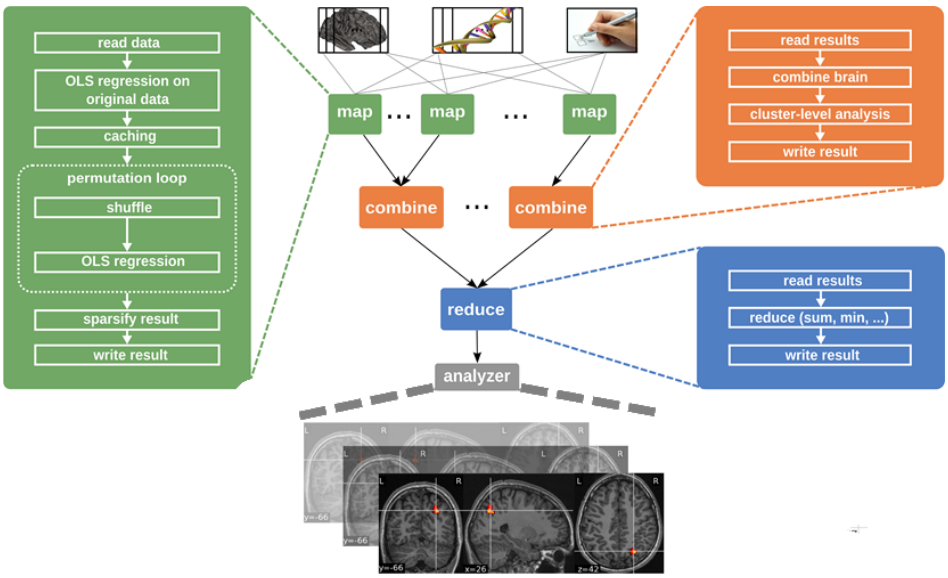


Figure 6.2: The A-Brain application as a MapReduce process

6.2 Towards a MapReduce Architectural Solution

The workload estimation for A-Brain clearly shows that the application needs to be run in parallel on a large set of resources in order to achieve reasonable execution timespans. Cloud computing platforms, such as Microsoft Azure, have the potential to offer the underlying infrastructure on which such a scientific application can be executed at a large-scale. Nevertheless, such computations can potentially be carried out also on other platforms such as supercomputers, grids or in-house clusters. However, our goal is to understand to what extent the cloud computing model can be used to drive scientific discovery. Being able to use the cloud for science will reduce the overall costs by moving from own managed infrastructures to a flexible pay-as-you-go model for the compute resources which can be scaled on-demand, while the administration overhead of owning the infrastructure is outsourced to the provider, as detailed in Chapter 3.

The MapReduce programming model has arisen as a very effective approach to perform high-performance data-intensive processing over very large distributed systems such as the clouds. This scalability property of MapReduce makes this compute paradigm an interesting choice for executing the A-Brain workload on the cloud. However, alongside with scaling the computation, an efficient storage solution also needs to be set up in order to enable the execution framework to harness the power of the cloud fully, easily and effectively. This is a particular challenge on which we focused next in Chapter 7.

Our first step was to parallelize the A-Brain application using the MapReduce paradigm as shown in Figure 6.2. Each Mapper takes the same initial data set, i.e., the neuro-images and the genes to compare against. It shuffles it to increase the statistical precision and performs the regression to assert the magnitude of the correlation. In terms of computation, the regression phase represents a series of matrix operations that generate a matrix of p-values. These matrices represent the intermediate data of the MapReduce process. In the case of the robust machine-learning analysis, the difference from the plain univariate method is that the

Map phase yields the prediction score for an image phenotype and tests the permutation under the null hypothesis to estimate the significance (i.e., p-value).

The Reduce phase collects the intermediate results, computes the statistic distribution and corrects the p-values and filters them. In the later step of the computation, only the correlations with a p-value higher than a specified threshold are considered relevant and kept. The Reduce operation is commutative and associative. An important constraint of the application is the requirement to reduce everything to a unique final result. This constitutes an important challenge for the classical MapReduce model that outputs as many final results as the number of reducers. We address these such challenges in Chapter 8.

Our initial empirical evaluations showed that one core in Azure is able to perform $\sim 1.47 \times 10^6$ associations per second (the small speed reduction is due to the virtualization of the cloud nodes). Our goal is to use the maximal number of cores that can be allocated for a deployment ($\sim$350), which would reduce the total computation time of A-Brain univariate analysis from several years to a few days (i.e., $\sim \frac{associations}{cores \times algorithm\_cloudspeed}$). Regarding the robust machine-learning analysis, the scale of the computation would entail an execution timespan of months for a single run. This estimation shows that executing this statistical robust A-Brain analysis at the scale of a single deployment will not be sufficient. Therefore, we decided to extend the computation for this scenario across several deployments in order to reach 1000 cores and reduce the time of the analysis down to about two weeks.

6.3 Discussion

The A-Brain analysis is representative of a large class of scientific applications that split a large initial domain into subdomains, each managed by a dedicated process (e.g., bio-informatics, image processing, weather simulations, etc.). The processes compute various metrics for all the components of their subdomain. These intermediate values are then exchanged, partially or totally, among the processes to perform some associative reductions in order to produce a single result (e.g., filtering, minimum/maximum, selection, etc.). Therefore, the approaches that we device in the context of A-Brain, presented in Chapters 7 and 8, have the potential to support other scientific applications migrated to the cloud. It also arguments in favor of our choice of using this application for validating the proposed solutions. Moreover, the A-Brain challenging demands for computation, data and scale, enable us to thoroughly evaluate the cloud support for Big Data applications and to identify the primary issues of executing scientific applications on clouds. Such an analysis, the bio-informatic results of A-Brain and the lessons learned are presented in Chapter 9, which serves as a requirements roadmap for the Part III of the thesis.

Chapter 7

TomusBlobs: Leveraging Locality for MapReduce Applications on Azure Cloud

Contents

7.1 TomusBlobs: Federating Virtual Disks for a Communication Efficient Storage 56

7.2 Leveraging Virtual Disks for Efficient MapReduce Processing 58

7.3 Validation and Experimental Evaluation 61

7.4 Extending TomusBlobs for Efficient Workflow File Management 65

7.5 Validation and Experimental Evaluation for Workflows 71

7.6 Discussion 73

This chapter develops the contributions published in the following papers:

- *TomusBlobs: Towards Communication-Efficient Storage for MapReduce Applications in Azure.* Radu Tudoran, Alexandru Costan, Gabriel Antoniu, and Hakan Soncu. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2012), Ottawa, Canada, May 2012, pp. 427-434.
- *Adaptive file management for scientific workflows on the Azure cloud.* Radu Tudoran, Alexandru Costan, Ramin Rezai Rad, Goetz Brasche, and Gabriel Antoniu. In the proceeding of the IEEE 2013 BigData conference (BigData 2013), Santa Clara, US, October 2013, pp. 273-281.
- *TomusBlobs: Scalable Data-intensive Processing on Azure Clouds* Alexandru Costan, Radu Tudoran, Gabriel Antoniu and Goetz Brasche. Journal of Concurrency and Computation: Practice and Experience 2013.

One missing facility that limits a larger adoption of clouds for scientific computing is data management, due to the lack of specific support for data-intensive scientific workflows. Currently, workflow data handling in the clouds is achieved using either some application-specific overlays that map the output of one task to the input of another in a pipeline fashion, or, more recently, leveraging the MapReduce programming model (e.g., Amazon Elastic MapReduce [8], Hadoop on Azure - HDInsight [86]). Such applications need a high-performance storage system that enable VMs to access shared data concurrently. However, today's reference commercial clouds only provide object stores such as S3 or Azure Blobs accessed through high-latency REST (HTTP) interfaces. Furthermore, situations may arise where applications might need to change the way data is managed in order to adapt to the actual access method (files vs. objects) [35].

The need for efficient storage for data-intensive workloads. A first approach for managing data would consist in relying on such public cloud object stores in the way the application would use a more traditional parallel file system. However, in today's cloud architectures, computational nodes are separate from the storage nodes and communication between the two exhibits a high latency due to the aforementioned data access protocols. Additionally, as these services primarily target storage, they only support data transfer as a side-effect, which means that they do not enable transfers between arbitrary VMs without intermediary storing the data. Moreover, users need to pay for storing and moving data in/out of these repositories in addition to the cost of leasing the VMs. Cloud providers recently introduced the option of attaching the cloud storage as virtual volumes to the compute nodes: Amazon EBS [7] or Azure Drives [17]. Besides being subject to the same high latencies as the default storage access, this option also introduces scalability and sharing limitations as only one VM can mount at a time such a volume.

An alternative to the cloud storage would be to deploy a parallel file system on the compute nodes, in order to exploit data locality when storing and transferring workflow data. Distributed storage solutions such as Gfarm [136] were deployed in a compute cloud — Eucalyptus, but operate in the host OS of the physical node in order to store the data in the local storage disks of the machine. This approach requires to specifically modify and

extend the cloud middleware which works in open-source IaaS clouds running on private infrastructures, but is not feasible in public commercial clouds. In fact, most file systems need special configuration or handling to get them to work in a virtualized environment, while others, as exemplified previously, cannot be executed at all, since they require kernel modifications which are not allowed by most cloud providers [103, 104]. When working at Platform-as-a-Service level (e.g., Microsoft Azure), users face additional challenges, making it difficult to set up even existing general-purpose runtime within the VM instances: there is no possibility to deploy a parallel filesystem like HDFS [85] or PVFS [81].

Cloud processing options. Besides efficient storage, data-intensive applications also need appropriate distributed computing frameworks, as presented in Chapter 5, to harness the power of clouds easily and effectively. However, options are rather limited on today's commercial clouds. On Microsoft's Azure cloud, there is little support for parallel programming frameworks: no MPI nor Dryad [100]. Yet, a MapReduce runtime, called AzureMapReduce [77], was proposed, built on top of the Azure BLOBs for data storage and on the Azure roles model of the VM instances (Web Role/Worker Role) for computations. However, the architecture of this system involves costly accesses from VMs to BLOB storage, whose efficiency is not satisfactory, as previously explained. Hadoop was ported to Azure as a service, called HDInsight [86], only recently and at a later time than our approach. Users are in charge of handling the data and setting their task to be run, but have no control on tuning or modifying the service (i.e., it is used as a black box). On the other hand, the major part of scientific applications do not fit the MapReduce model and require a more general data orchestration. In this direction, several workflow management systems were proposed for applications running on clouds. e-Science Central [91] and the Generic Worker [153], both built on top of the cloud storage, enable scientists to harness vast amounts of compute power by running the applications in batches.Other systems like Pegasus [2] rely on a peer-to-peer file manager, when deployed on Amazon EC2, but use basic transfers between the VMs with very low performance.

Our approach in a nutshell. To address these issues for managing data in the clouds, we propose an architecture for *concurrency-optimized, PaaS-level cloud storage leveraging virtual disks*, called *TomusBlobs*. For an application consisting of a large set of VMs, it federates the local disks of those VMs into a globally-shared data store. Hence, applications directly use the local disk of the VM instance to share input files and save the output files or intermediate data. As demonstrated by the results presented in this chapter, this approach increases the throughput more than 2 times over remote cloud storage. Moreover, the benefits of the TomusBlobs approach were validated in the context of MapReduce, by building an Azure prototype which implements this computation paradigm and uses the proposed storage approach as data management back-end. It reduces the timespan of executing a scientific application by up to 50 %, as shown in Section 7.3. Furthermore, we extend this data management solution also for general workflow to leverage data locality for direct file transfers between the compute nodes. We rely on the observation that workflows generate a set of common data access patterns that our solution exploits to self-adapt and to select the most adequate transfer protocol, which speeds up transfers with a factor up to 2 over current data management options, as discussed in Section 7.5.

7.1 TomusBlobs: Federating Virtual Disks for a Communication Efficient Storage

This section introduces the TomusBlobs approach for federating the virtual disks of VMs. The system addresses the main requirements of data intensive applications, detailed in Section 4.2, by providing low-latency data storage optimized for concurrency. We start from the observation that the disks locally attached to the VMs, with storage capacities of hundreds of GBs available at no extra cost, are not exploited to their full potential in many cloud deployments. Therefore, we propose to aggregate parts of the storage space from the virtual disks in a shared common pool that is managed in a distributed fashion. This pool is used to store application-level data. In order to balance the load and thus to enable scalability, data is stored in a striped fashion, i.e. split into small chunks that are evenly distributed among the local disks of the storage. Each chunk is replicated on multiple local disks in order to survive failures. With this approach, read and write access performance under concurrency is greatly enhanced, as the global I/O workload is evenly distributed among the local disks. Furthermore, this scheme reduces latencies by enabling data locality and has a potential for high scalability, as a growing number of VMs automatically leads to a larger storage system.

TomusBlobs is designed accordingly to the following set of design principles. These principles were selected such that they comply with the otherwise typically contradicting constraints of cloud providers and scientific high-performance computing.

Data locality. Accessing data from remote locations increases the cost of processing data (both financially and compute-time wise). Yet, in today's cloud model, the computation regularly uses the cloud storage for I/O while the locally and freely available virtual disks from each VM remain largely unused. This applies even for intermediate results of large-scale scientific processing, like MapReduce or general workflows. This reduces the overall execution performance. Thus, our goal is to leverage this free local VM storage space by aggregating and managing it in a distributed fashion and making it available to the applications. Additionally, the overall cost is reduce as we decrease the usage of the otherwise payable cloud storage.

No modification of the cloud middleware. Our approach targets the commercial public clouds. It is therefore mandatory that its building blocks do not require any special or elevated privileges. As our data management approach is deployed inside the VMs, the cloud middleware is not altered in anyway. This is a key difference from the previous attempts to harvest the storage physical disks of the compute nodes. These attempts imposed modifications to the cloud infrastructure, so that they only worked with open source cloud kits. Thus, our solution is suitable for both public and private clouds. It addresses standard cloud users such as scientists which do not possess the skills or permission to configure and manage the cloud middleware toolkit.

Loose coupling between storage and applications. The TomusBlobs cloud data management is mainly targeted at (but not limited to) large-scale scientific applications executed like the MapReduce computations. Therefore, we propose a modular, stub-based architecture, which can easily be adapted to other processing paradigms, particularly data-intensive workflows as illustrated in Section 7.4.

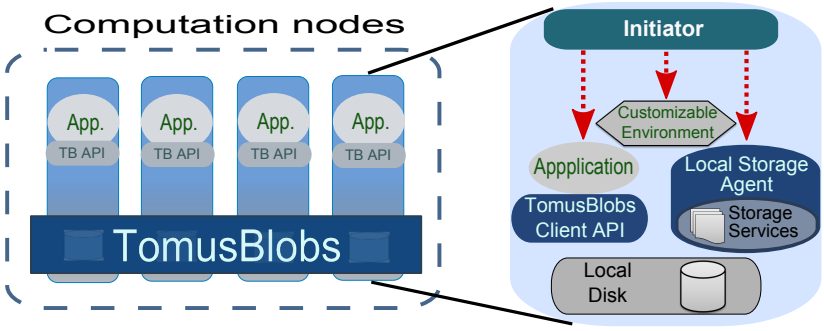


Figure 7.1: The architecture of TomusBlobs (TB).

No centralized control. When scaling computation or storage to a large number of resources, a centralized control of the data flow rapidly becomes a bottleneck. This is a key aspect for cloud infrastructures which lease commodity hardware which is subject to failures and where the high-bandwidth is not typically the norm. Therefore, we choose to address the coordination between the components in a distributed fashion, without any centralized supervision.

Building on these design principles that exploit data locality, we designed TomusBlobs, a system for concurrency-optimized PaaS-level cloud storage.The system relies on the local disk of the VM instance directly in order to share input files and save the output files or intermediate data. This approach requires no changes to the application nor to the cloud middleware. Furthermore, it does not use any additional resources from the cloud, as the virtual disks are implicitly available to the user for storage, without any additional costs. We implemented this approach in the Microsoft Azure cloud platform. The architecture of TomusBlobs consists of three loosely-coupled components presented in Figure 7.1.

The Initiator component is specific for each cloud platform. It has the role to deploy, setup and launch in a transparent way the data management system. Moreover, it takes the role of customizing the scientific environment that is usually required by the applications. It exposes a generic stub that can be easily implemented and customized for any cloud API. It does so, as most storage solutions require some prior knowledge about the underlying infrastructure when they are started, e.g., IPs of the VMs. By interacting with the cloud middleware, it acquires all the necessary information enabling the system to be self-configurable. The Initiator running within each cloud node is in charge of assigning roles, differentiating between instances by hashing an ID from a set of parameters (name, IP, etc.). Additionally, it is used to setup the system on the deployment, based on user policies, e.g., the number and storage entities to run on separate nodes. Finally, the Initiator supports the system's elasticity, being able to scale up and down the computing platform at runtime by integrating the newly active nodes in the system or by seamlessly discarding the deactivated ones.

The Local Storage Agent has the role of aggregating the virtual disks into a uniform shared storage, which is exposed to applications. It is generic as it does not depend on any specific storage solution. Any distributed file system capable to be deployed and executed in a cloud environment (and not changing the cloud middleware) can be used as

a storage backend. This implies contextualizing the Storage Agents with respect to the architecture of the adopted solution to manage its "storage services", i.e., the composing entities of the solution (disk and metadata managers, transfer modules etc.). For the proof of concept we opted for two distinct approaches:

- Integrating an existing solution called BlobSeer [132], described in Chapter 5. We use this for low level object-based storage, for computation scenarios such as MapReduce.
- A new file management system that we designed for workflows, further described in Section 7.4. Building a new solution was motivated by the fact that there are no data management tools specialized for workflows on clouds.

The Client API represent the layer through which the storage is uniformly viewed and accessed by the applications. Data manipulation is supported transparently through a set of primitives, allowing applications to access the BLOBs with full concurrency, even if all access the same BLOB. The interface is similar to the ones of commercial public clouds (Azure BLOBs, Amazon S3): one can get data from the system (*READ*), update it by writing a specific range within the BLOB (*WRITE*) or add new data to existing BLOBs (*APPEND*). The Client API is implemented such that it hides all calls and interactions between the storage entities when performing the exposed operations.

The local storage of VMs on which we rely consists of virtual block-based storage devices that provide access to the physical storage of the compute nodes, unlike the attached storage volumes (e.g., Amazon EBS or Azure drives) which link the VM with the remote cloud storage.The virtual disks appear as devices to the virtual machine and can be formatted and accessed as if they were physical devices. However, they can hold data only for the lifetime of the VM, and in this sense they are an ephemeral storage option. After the VM is terminated, they are cleared. Hence, it is not possible to use them for long-term storage since this would mean leasing the computation nodes for long periods. Instead, we have designed a simple checkpoint mechanism that can backup data from the TomusBlobs to the persistent Azure BLOBs as a background job, privileging the periods with little / no network transfers and remaining non-intrusive (it adds a 4 % computational overhead when a transfer is performed). The backups follow simple policies to maintain consistency by either making independent snapshots of the system each time or by maintaining only one version.

7.2 TomusBlobs-MapReduce: Leveraging Virtual Disks for Efficient MapReduce Processing

In order to demonstrate and validate the efficiency of the TomusBlobs storage solution in the context of cloud data processing, we built a prototype MapReduce framework for the Azure cloud, called TomusBlobs-MapReduce. The proposed framework relies on TomusBlobs to store input, intermediate and final results, allying data locality with the elastic computational power of the cloud. With the storage and computation in the same virtualized space, data transfer, protection, confidentiality and security are enhanced, benefiting from the usage of the local storage instead of the remote Azure Storage. From the job scheduling perspective and for the coordination between the system entities, we opted for a light and non-intrusive mechanism built on top of the cloud queues (i.e., Azure Queues).

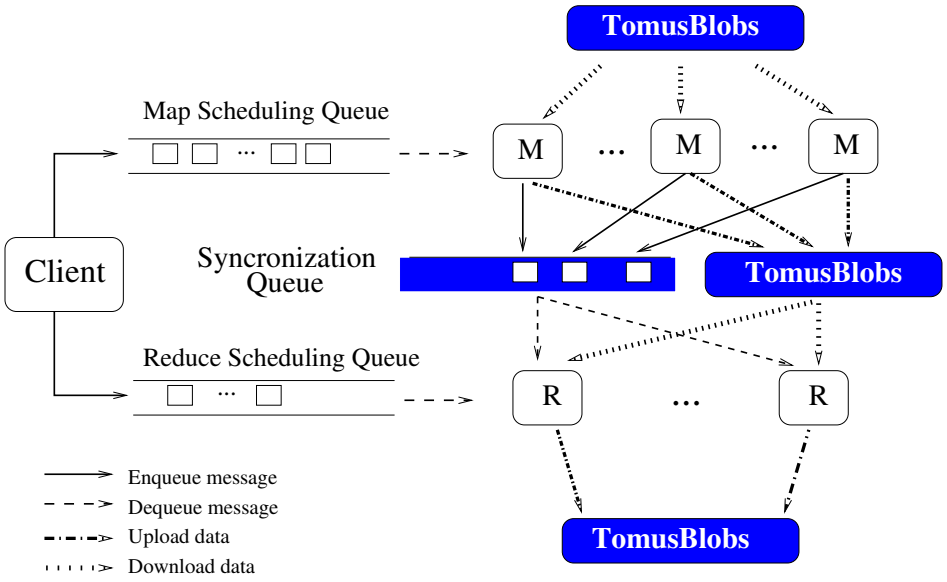


Figure 7.2: The TomusBlobs-MapReduce architecture.

Why a new MapReduce framework? The motivation for creating such an engine is two-folded. On the one hand there is no MapReduce framework that enables to properly exploit data locality in the context of PaaS clouds: AzureMapReduce uses the remote azure Blobs, while Hadoop on Azure was introduced at a later time as a black-box service. On the other hand, unlike the MapReduce cloud-provided services such as HDInsight and Amazon MapReduce, which require to outsource the computation and data to these services, our framework can be deployed on users compute nodes, together with the corresponding data and potentially other compute services. Moreover, by creating a user-customizable framework, the MapReduce processing pipeline can be extended and tune for computation flows beyond the default model. Obtaining this property for the system, which allowed us to experiment with different declinations of the MapReduce paradigm, was another argument in favor of building our own framework.

The architecture of the TomusBlobs–MapReduce solution that we proposed is depicted on Figure 7.2. User clients are provided with a web or console-based front-end through which their jobs are submitted. The scheduling of the MapReduce jobs on the compute nodes is performed through small messages, which encode the actual task description and the related metadata (e.g., input/output location, naming scheme, etc.). These messages are disseminated towards the compute entities using 3 queues: the *Map Scheduling Queue* is used to submit jobs towards Mappers, the *Reduce Scheduling Queue* is used to submit jobs towards Reducers and the *Synchronization Queue* is used by the Mappers to notify the Reducers about map job completion. The advantage of this scheduling mechanism is that it has a light footprint on the actual compute resources, making it non-intrusive and isolated from potential failures that might appear in the compute process. The Mappers and the Reducers are implemented as Azure Worker Roles that execute the user-provided functions.

The *Client* is a web-service front-end deployed on a the Azure Web Role, i.e., the PaaS Azure compute option specialized for hosting web services. It offers users the tools to specify and configure the MapReduce workflow and to submit it for execution. Based on these spec-

ifications, the client component creates the appropriate message description for the Mappers and Reducers. These are then scheduled and submitted via the corresponding queue. The messages are small in size (in the order of several KBs). They contain the scheduling identifier of the destination Mapper or Reducer, used to map the tasks to the resources and metadata information about the task itself, e.g., data location in the storage, naming scheme, job and data ownership.

The *Mappers*, marked with "M" in Figure 7.2, are deployed in Azure Worker Roles, i.e., the default compute PaaS Azure option. Based on the specified policies which are managed by the Initiator component of TomusBlobs, a pool of Mappers is created. They are regularly polling the Map Scheduling Queue in order to retrieve messages with job description submitted by the user through the client front-end. Similarly, the *Reducers*, denoted "R" on Figure 7.2, are run on Azure Worker Roles, with a pool of such entities created at start time. The reducers first poll the queue through which the client front-end submits the job description. After a task is being assigned to them, the reducers start listening to the synchronization queue. They wait for messages sent from the Mappers, notifying the end of a map computation and the availability of intermediate data. When such notifications appear the reducers dequeue the message and fetch the corresponding intermediate data from the TomusBlobs storage.

The *Azure Queues*, described in Chapter 3, are used as a scheduling mechanism by TomusBlobs-MapReduce. We rely on the visibility timeout of the queues to guarantee that a submitted message will not be lost and will be eventually executed by a Mapper or a Reducer. A message which is read from the queue is not deleted, but instead hidden until an explicit delete is received. Our framework uses this mechanism to explicitly delete the jobs marked as hidden only after a successful processing. If no such confirmation arrives, the message will become visible again in the queue, after the visibility timeout. Duplication of job execution is possible. Nevertheless, by using a uniform naming schema for the job outputs, only one instance of the result will be stored and considered further on in the computation, preventing in this way inconsistent states of the output result. Therefore, the scheduling process of our system is protected from unexpected node crashes as well as slow nodes. Hence, the fault tolerance is addressed both at data level, using the replication support of TomusBlobs, and at processing level, by using the properties of Azure Queues.

By building on the TomusBlobs approach, we were able to provide several functionalities needed by scientific applications which are now available using the MapReduce model. While Hadoop does not support runtime elasticity (working with a fixed number of mappers and reducers), our solution seamlessly supports scaling up and down the number of the processing entities, as follows. The MapReduce engine is deployed and configured using the mechanism provided by TomusBlobs (i.e., the Initiator). Thus, when scaling the deployment, the Initiator is also able to update the parameters of the MapReduce framework. The parameter value holding the number of reducers, used for hashing the map results, is dynamically updated which enables elastic scaling. Moreover, the simple scheduling schema combined with the flexibility of the system enables users to easily extend and tune the MapReduce processing pipeline. One can easily add modules and alter the processing paradigm with new processing stages and computation flows beyond the default model. Such an extension, which addresses the specific constraint of bio-informatics applications for a unique result is Map-IterativeReduce, detailed in Section 8.1.

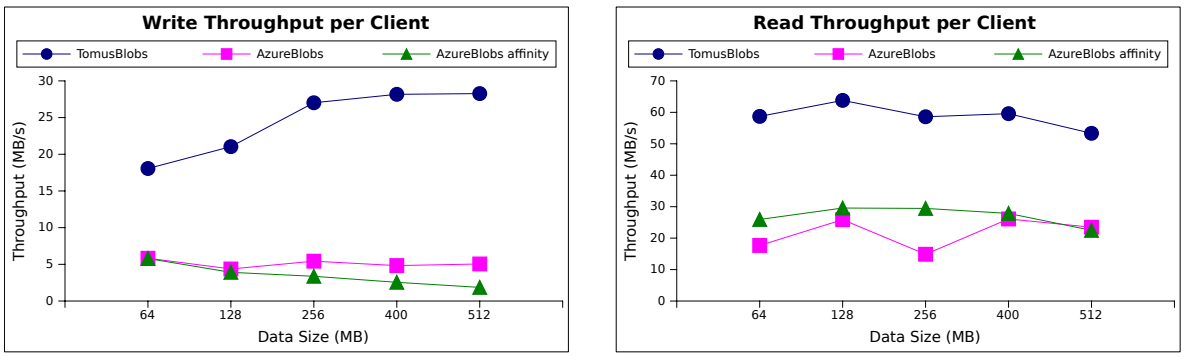


Figure 7.3: Storage write and read throughput with respect to data size for a client

7.3 Validation and Experimental Evaluation

Next, we evaluate the benefits of the TomusBlobs approach in synthetic settings and using A-Brain application. The experiments were performed on the Azure cloud using 100 Small (1 CPU cores, 1.75 GB memory, 225 GB local disk) Azure VM Worker Roles. For a better accuracy, we repeated the measurements hundreds of times at various moments of the day (morning, afternoon, night) and in two geographically distributed sites: North Europe and North-Central US data centers. The evaluation focuses on comparing the throughput of TomusBlobs against the default Azure BLOB storage and on the related costs. The BlobSeer system was configured to use 2 nodes for the Version and Provider managers, 20 nodes for the Metadata Providers and 78 nodes for the Data Providers.

7.3.1 Cloud Storage Evaluation: TomusBlobs vs. Cloud-Provided Storage Service in Synthetic Settings

The first series of experiments evaluate the throughput performance of our proposal in controlled synthetic settings. The goal of these experiments is to assess the benefits that can be expected when using a low-latency cloud storage service such as TomusBlobs for data intensive applications. For this, we have implemented a set of micro-benchmarks that write and read data in Azure and measured the achieved throughput as more and more concurrent clients access the storage system. Writes and reads are done through the Client API for TomusBlobs and through the RESTful HTTP-based interface for Azure BLOBs. We have focused on the following access patterns exhibited by highly-parallel applications.

Scenario 1: single reader/writer, single data. We first measured the throughput achieved when a single client performs a set of operations on a data set whose size is gradually increased. This scenario is relevant for the applications where most of the time each process manipulates its own data set independently of other processes (e.g., simulations, where the domain is typically distributed among processes that analyze it and save from time to time the local results). The throughput for read and write operations for TomusBlobs and Azure BLOBs are shown in Figure 7.3. We report the average of the measurements of the two for which we observed a standard deviation of $\sim$12 for reads and $\sim$2 for writes. The Azure

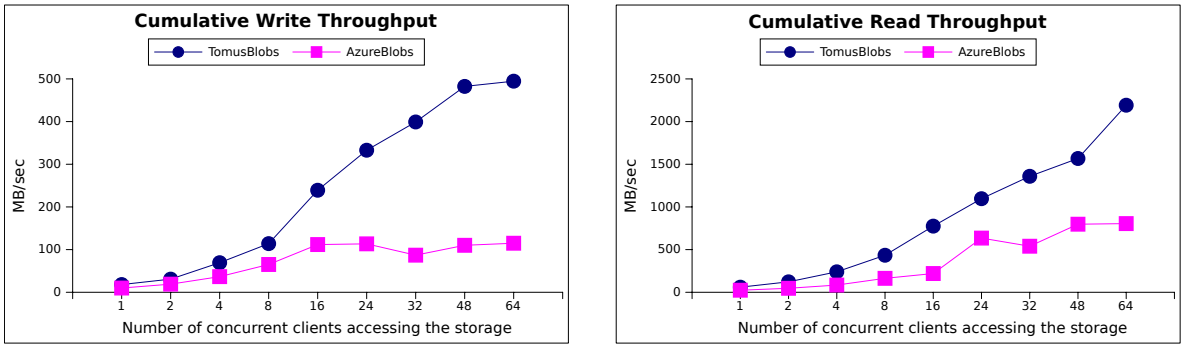


Figure 7.4: Write/Read throughput per client under concurrent access

BLOB measurements are consistent with the ones reported by Hill et al. [93] and [30]. The evaluation is done by transferring with a single operation the data from the application memory to storage. Therefore, the size of the manipulated data is increased up to the limit of the application memory available in a Small VM. TomusBlobs achieves a significantly higher throughput than Azure BLOBs (approximately 2 times higher), as a result of using the low-latency local disks. Also, the fact that they combine the workloads of many different users together to reduce storage usage [35] (which is not the case of TomusBlobs) penalize the performance of Azure BLOBs. Finally, another important factor that influences the observed I/O throughput in Azure is the concept of "affinity groups". It allows an increased proximity for the co-location of storage and hosted services within the same datacenter. Despite that this option can reduce latencies for the Azure storage, is can be observed that is significant less efficient than TomusBlobs, by more than 2 times, validating our proposal of collocating data in compute nodes.

Scenario 2: multiple readers/writers, single data. In the second series of experiments we have gradually increased the number of clients that perform the same operation concurrently and measure the aggregated throughput. For each given number of clients, varying from 1 to 65, we executed the experiment in two steps. First, all clients write concurrently 128 MB of random data from memory to the storage and in a second step they read it back.This scenario is relevant for applications where multiple clients concurrently read the input data or write the temporary or final results or process data in multiple phases (e.g., MPI, iterative computation). Another example for this pattern is the “map” phase of a MapReduce application, in which mappers read the input in order to parse the (key, value) pairs. The evaluation of the aggregated throughput for concurrent read and write operations is presented in Figure 7.4. The results are average values with a standard deviation of $\sim$15 MB/s for the reads and $\sim$ 5 MB/s for the writes. An upper limit for the performance can be observed for the cumulative throughput for an increase number of clients, especially for the writes. For TomusBlobs this is explained by the network saturation due to the total amount of data that is sent at a time. For Azure BLOBs the limitations are: the latency between the computation and storage nodes, the available bandwidth between them and the way the storage system handles multiple requests in parallel. TomusBlobs outperforms the Azure Storage, having a cumulative write throughput 4 times higher and a read throughput 3 times higher for more than 30 concurrent clients, leveraging both the parallel data access scheme of the underlying

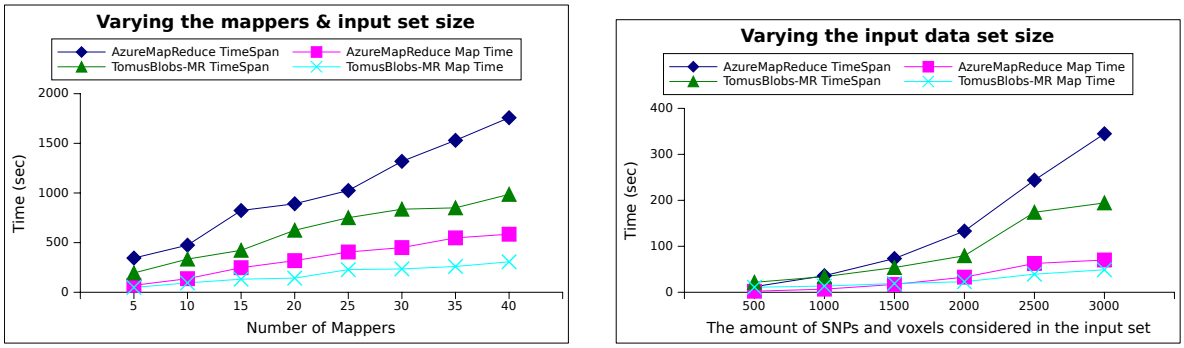


Figure 7.5: a) Left: Application execution time and transfer time for intermediate data when the size of data genetic and neuro-imaging data is varied and the number of Mappers is kept constant. b) Right: Application execution time and transfer time for intermediate data with AzureMapReduce and TomusBlobs-MapReduce when the number of map jobs and the size of the data is increased

storage backend and the data locality.

7.3.2 Initial Experimentation with the A-Brain Application

In a second phase, our goal was to assess the impact of TomusBlobs in the context of MapReduce using the A-Brain scientific application, discussed in Section 6. The comparison is done between the TomusBlobs-MapReduce engine we have build and the AzureMapReduce framework, which relies on Azure BLOBs for storage. A first series of experiments focuses on the total execution time when the data size and the number of Mappers are progressively increased, while in a second phase we measured the aggregated throughput under high concurrency.

Completion time: increasing the data size. Figure 7.5 a) presents the completion time of the MapReduce computation for the two frameworks.The number of Mappers (5) and Reducers (1) was fixed while the size of the input set was increased from 30 MB up to 2 GB; conceptually this means that more brain regions (voxels) and genes are analyzed.When using TomusBlobs-MapReduce, the workload is processed up to 40 % faster than when using AzureMapReduce, which in turn is shown to have similar performances with Hadoop or Amazon MapReduce [77]. To better evaluate the impact of the the increased throughput brought by TomusBlobs, we measured and report in the same figure, also the time for transferring data between mappers and reducer (through the storage system). These results show that write-intensive maps and read-intensive reducers can gained from a better support for data management reducing the overall execution time.

Scalability: increasing the number of Mappers. Figure 7.5 b) presents the execution timespan of A-Brain when keeping the size of the input data constant - 1 GB (the overall amount of data is approximately 40 GB) and increasing the number of Mappers. In practice, this

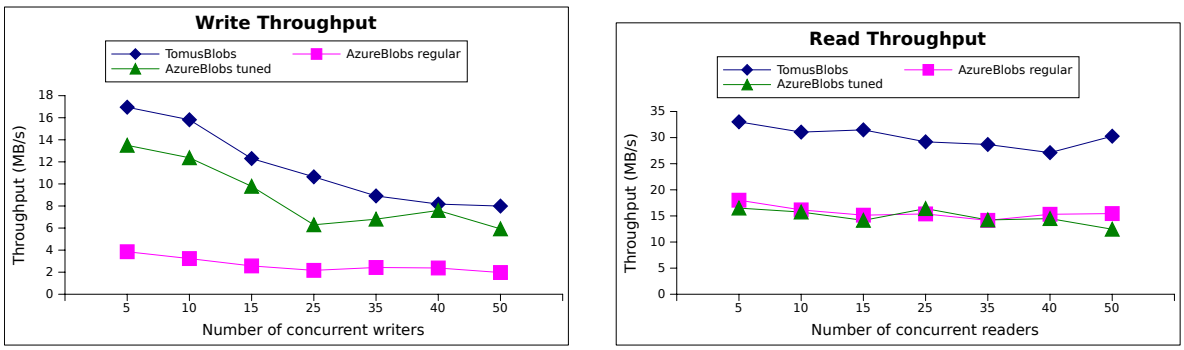


Figure 7.6: Storage read/write throughput in the context of MapReduce application

means that more shuffles are performed (each Mapper performs a shuffle as detailed in Section 6) in parallel, thus increasing the precision of the univariate analysis at the cost of extra computation. Thus, unlike a typical MapReduce, by increasing the number of map jobs, we do not just increase the degree of parallelism for analyzing a fix data set, but rather the workload increases proportionally.Each Mapper generates its own intermediate results, having a constant size given the initial data. Increasing the number of Mappers is equivalent to generating more intermediate results that must be processed in the Reduce phase. Data proximity significantly reduces the completion time with TomusBlobs-MapReduce, especially for a larger number of Mappers which can read and write in parallel from the virtual disks. It is not the case for AzureMapReduce for which the latencies induced by the remote storage increase completion time with up to 55 %. As before, by isolating the processing times for the intermediate data (i.e., Map time in Figure 7.5), we notice TomusBlobs supports efficient data handling under heavy concurrency and demonstrating it successfully brings data locality in the context MapReduce on PaaS clouds.

Throughput: evaluating the I/O pressure. The previous experiments shown that the way in which the intermediate data is handled is critical for the execution time of the A-Brain univariate analysis. Therefore, we proceeded to a more in-depth analysis of this stage by testing the read and write patterns of reducers and mappers (each of them processing 128 MB of data) against different storage solutions: TomusBlobs, AzureBlobs and AzureBlobs with a configuration for multi-threading handling of reads/writes, with the number of threads set to 8 (the maximum value allowed by Azure). The results are presented in Figure 7.6 through average values with a standard deviation for the reads of 4.53 for TomusBlobs and 1.39 for both Azure BLOBs options, while writes had 4.4 for TomusBlobs, 2.34 and 0.48 for the two write types to Azure BLOBs. The decreasing tendency of the throughput in all cases is explained by the network saturation. The evaluation shows that even when we use CPU cycles for faster data transfers, the cost of accessing a remote public storage is higher than with our approach.

7.3.3 A Cost Analysis for Executing Scientific Applications on the Cloud

We are interested to assess the cost of running scientific applications, such as A-Brain, in a public cloud. To break down the cost, we start by dividing the overall workload into 2 parts:

computation and transferring data. For expressing the corresponding costs analytically, we assume that A is the overall computation workload to be done, f_n is the fraction of the data analyzed per second if we use N computing nodes and D is the total amount of data manipulated. For the particular case of the univariate analysis of the neuro-imaging data, A represents the number of associations (significant links) to be tested between genes and brain locations, while f_n represents the number of associations per second performed by the N machines. Hence the two components of the cost are:

$$cost_{computation} = \frac{A}{f_n} \times N \times c_{VM}, \text{ where } c_{VM} \text{ - cost of a machine compute hour}$$

$$cost_{data} = D \times (c_s + c_t), \text{ with } c_s \text{ - cost for storing and } c_t \text{ - cost for transfer}$$

Computing the cost for storing the data is not straightforward since it is based on monthly rate fees, but it is computed based on hourly averages of the amount of data stored in a day. For applications where most of the data are transient (as for A-Brain), the data will be stored only until it is processed and discarded afterwards. Therefore, we will compute the data cost as the overall amount of data multiplied by the cost to store data per month normalized by the workflow timespan:

$$cost_{data} = D \times cost_{hour} \frac{\text{Nr\_of\_hours}}{31 \times 24}$$

Based on these formulas we computed the storage cost of running A-Brain on 200 cores deployment.This amount of cores were required to execute the univariate analysis of A-Brain in order to show the sensitivity of the technique to the data outliers [45]. The average computation speed of the algorithm at this scale is 2.94×10^8 associations per second while the data space reaches $\sim$10 TB. We consider the price of one hour of computation of 0.08 euros and a cost for storing 1 GB for 1 month of 0.11 euros [19]. We obtain:

$$cost_{total} = \frac{2.5 \times 10^{14}}{2.94 \times 10^8} \times 200 \times 0.08 + 10 \times 1024 \times 0.11 \times \frac{\frac{2.5 \times 10^{14}}{2.94 \times 10^8}}{31 \times 24} = 4133 \; euros$$

This provides an estimate of the cost to process scientific data in commercial clouds using the cloud compute and storage options. However, using the storage and data processing platforms that we proposed, this cost can be significantly reduced. Considering the average speedup of 25 % brought by TomusBlobs-MapReduce and storing data locally with TomusBlobs instead of Azure BLOBs, the cost is decreased with more than 30 %, down to 2832 euros. These results show that the benefits of TomusBlobs are not limited just to accelerate the process of scientific discovery but also to reduce the cost of it.

7.4 Extending TomusBlobs for Efficient Workflow File Management

Many scientific computations cannot be reduced to the MapReduce paradigm. Therefore, several workflow frameworks have been ported to the clouds to enable more general processing. Unfortunately, they were not designed to leverage the cloud infrastructures for

handling large data volumes. The major bottleneck, just like like for MapReduce processing, is the lack of support for efficiently handling and transferring data between jobs. Thus, supporting data-intensive workflows on clouds requires to adapt the workflows engines to the virtualized cloud environment of commodity compute nodes and to optimize data transfers and placement to provide a reasonable time to solution. Our approach is to apply the principles of TomusBlobs in the context of cloud workflow processing: data locality, federating virtual disks, no modifications to the cloud middleware. For understanding which are the best options for this, we refer to a study on 27 real-life workflow applications [66] from several domains: bio-informatics, business logic, simulations, etc. This survey reveals that workflows: 1) have common *data patterns* as shown also in [167]: broadcast, pipeline, gather, reduce, scatter; 2) are composed of batch jobs, i.e., stand-alone executable algorithms, which can be run with well-defined data passing schema; 3) use uniquely identifiable files as inputs and outputs of the batch jobs; and 4) write usually only once the input and output files.

We argue that keeping the data produced by the batch jobs in the local disks of the VMs is a good option considering that these files are usually temporary — they must exist only to be passed from the job that produced them to the one it will further process them. Taking into account that the files are written once, using as the storage entity of TomusBlobs a concurrency-write optimized backend such as Blobseer, which splits files and distributes them across nodes, is not the best choice for this scenario. Ergo, building on the observations of the aforementioned study, we designed a new approach for managing workflow data by making the files from the virtual disk of each compute node available directly to all nodes within the deployment. Caching the data where it is produced and transferring it directly to where it is needed reduces the time for data manipulation and minimizes the workflow makespan. This approach extends the initial set of design principles of TomusBlobs (Section 7.1) with two new ones.

Storage hierarchy. A hierarchy for data handling should be constructed, comprising: in-memory storage at the top, local disks, shared VM-based storage and finally the remote cloud storage at the bottom. A decreasing level in the hierarchy reduces the performance but tends to raise the capacity and costs. Indeed, as opposed to a classical computer architecture, the costs tend to increase towards the base of the hierarchy as the remote storage comes at an extra-cost while the local resources are available for free, becoming available when one rents compute cycles. Files are moved up and down the storage hierarchy via stage-in/out and migrate operations, respectively, based upon data access patterns, resource availability and user requests.

Integrate multiple transfer methodologies. Adopting several ways to perform file transfers such as peer-to-peer based, direct or parallel transfers between compute nodes, and dynamically selecting between them at runtime should be available, in order to increase the performance of handling data. More, the decision process of switching between the transfer options can be mapped to the workflow specific data access patterns and context information. This opens the avenue for customization, with users being able to easily add their own transfer modules, which can leverages the application semantics.

The architecture which extends the Local Storage Agent of TomusBlobs for workflow file management is depicted on Figure 7.7. The input and output files of the workflow batch

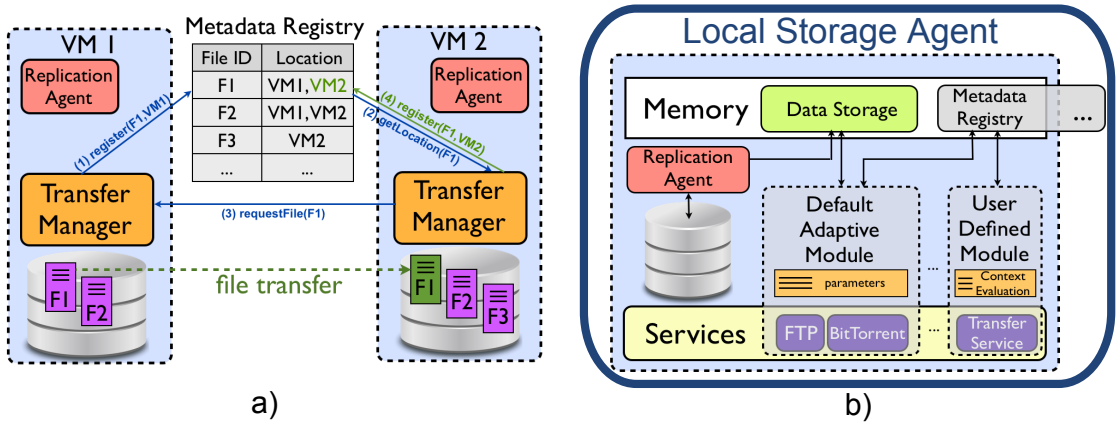


Figure 7.7: The architecture of the TomusBlobs File Management System and the optional replication module. Subfigure a) presents the operations for transferring files between VMs: upload (1), download (2, 3, 4) and b) shows the system components within a virtual machine.

jobs are stored on the local disks of the VMs. File sharing between tasks is achieved by advertising file locations and transferring the file directly towards the destination, while dynamically switching between the transfer modules (Figure 7.7a). The data management has three components, which are instantiated on each compute node (Figure 7.7 b).

The Metadata Registry holds the file locations, i.e., maps files to VMs. The metadata is organized as an all-in-memory distributed hash-table that holds key-value pairs: *file ids* (e.g., name, user, sharing group, etc.) and *locations* (the information required by the transfer module to retrieve the file). In a general scenario, a concurrency handling mechanism for writes/updates would be needed for the Metadata Registry, usually provided at the expense of performance. However, as previously stated, in the context of file management for workflow execution, the files are produced and written by a single task and uniquely identified: there are no situations in which two tasks request the Metadata Registry to publish the same new file. Therefore, there is no need for strong verification mechanisms to detect and solve eventual duplication conflicts. Several implementation alternatives are available: in-memory databases, Azure Tables, Azure Caching [15]. For our implementation, we chose the latter as it easily allows to dedicate for caching a percentage of each VM's memory. Furthermore, our preliminary evaluations showed that the Azure Caching delivers better performances than the Azure Tables (10 times faster for small items) and has a low CPU consumption footprint (unlike a database).

The Transfer Manager component enables applications or workflow engines to share the files via a simple API (e.g., *uploads* and *downloads*). The upload operation is equivalent to advertising a file which is done by creating a record in the Metadata Registry. Hence, the cost of uploading is $O(1)$, independent of the data size, consisting only of the time to write the metadata. The download operation is done in two phases: the file information is retrieved from the Metadata Registry and then data is fetched from the VM which holds it, reducing the number of read and writes operations to one, as shown in Figure 7.7. Multiple options are available for performing a transfer and, as stated in the design principles, our proposal is to

integrate several of them. Thus, this component is designed in a modular fashion that makes it easy to plug-in different transfer back-ends, i.e., libraries corresponding to a data transfer technology. The solution most appropriate for a particular context is selected. This was implemented using the Management Extensibility Framework [124], a utility which allows the creation of lightweight extensible applications without prior configurations. Figure 7.7 b) shows a snapshot of the transfer modules deployed in a VM. Essentially, the system is composed of user-deployed and default-provided transfer modules and their service counter parts, available on each compute instance. The client applications or workflow engines will interact with the local Transfer Manager component to download and upload files.

The Replication Agent is an optional component designed to ensure fault tolerance and to balance the transfer load of multiple accesses to a file through replication. As such, it manages several replication strategies and policies within the file management system. A secondary role to the replication is to evaluate the potential transfer time reductions, brought by increasing the number of replicas. These gains in time are correlated with a storage cost schema that we propose. This method, described bellow, enables the system to determine dynamically the appropriate number of replicas for each transfer context based on user policies. The replication strategies can be further extended with other strategies (e.g., [36, 164]), in order to schedule the replica placement in agreement with the workflow engine. The Replication Agent is implemented as a service that runs as a background process on each compute VM. In order to decrease its intrusiveness, the data transfers are performed only when the network bandwidth of the corresponding VM is not saturated by the Transfer Manager. For the communication and coordination between agents, a message-passing protocol over the Azure Queue was built.

Selecting the transfer method. Users can deploy their own transfer modules by means of a straightforward API, that only requires to add an evaluation function for scoring the context. The score is computed by aggregating a set of weighted context parameters, e.g., number or size of files, replica count, resource load, data format, etc. The weights reflect the relevance of the current transfer module for each specific parameter. For instance a fast memory-to-memory data transfer protocol will favor transfers of many small files through higher weights for these parameters. The module with the best score is selected for each transfer, as shown in Algorithm 1, Lines 3–10. If no user modules are deployed or none fits the context, a default module is chosen adaptively, Algorithm 1, Lines 12–28. The default module selection strategy uses a set of parameters defined by users in an XML file such as size limits of files to be fitted in memory, replicas count, etc. The weighting of each parameter is rated by both clients, that is workflow engines, and the Replication Agent. The latter can in fact modify the transfer context by increasing the number of replicas if the transfer speedup obtained comes at a cost that fits the budget constraints. Currently, the selection is done between three transfer protocols that we provide within our framework.

In-Memory. For small files or for situations in which the leased infrastructure has enough memory capacity, keeping data in the memory across VMs becomes interesting. This option provides one of the fastest methods to handle data, boosting the performance especially for scatter and gather/reduce access patterns. The module is implemented

Algorithm 1 Context-based transfer module selection

```
1: procedure ADAPTIVETRANSFERMODULESELECTION
2:     TransferInfo = ClientAPI.getTransferDescription()
3:     for all module in UserDeployedModules do
4:         score = module.ContextEvaluation(TransferInfo)
5:         best_score = max(best_score, score)
6:     end for
7:             ▷ Assert if client-provided modules can be use for transfer otherwise use a default one
8:     if best_score > user_defined_threshold then
9:         TransferModule = BestModule;
10:    end if
11:
12:            ▷ Weight active (Client) and passive (Replicator) transfer parameters based on budget
13:    ReadAdaptiveParameters(UserDefinedParams_XML)
14:    (SizeW_Client, ReplicaW_Client) = ClientAPI.getRecomandedScore()
15:    (SizeW_Replicator, ReplicaW_Replicator) = Replicator.getRecomandedScore(CostRatio_User)
16:            ▷ Try to speedup future file transfers through replication within the budget constraints
17:    Replicator.updateContextForSpeedup( TransferInfo, CostRatio_User)
18:
19:            ▷ Select the default transfer module that best fits the context and the client constraints
20:    if Size_File × (SizeW_Client + SizeW_Replicator) < MemoryThreshold_User then
21:        TransferModule = InMemoryModule
22:    else
23:        if Replicas_File × (ReplicaW_Client + ReplicaW_Replicator < ReplicaThreshold_User then
24:            TransferModule = TorrentModule
25:        else
26:            TransferModule = DirectLinkModule
27:        end if
28:    end if
29:    Client.notify(TransferModule, TransferInfo)
30:    return TransferModule
31: end procedure
```

using Azure Caching by aggregating a percentage of VMs memory into a shared system, independent from Metadata Registry.

FTP. For large files, that need to be transferred from one machine to another, direct TCP-transfers are privileged. FTP seems a natural choice for interoperability reasons. The data access patterns that benefit most from this approach are pipeline and gather/reduce. The module is built using the open-source library [61], which we tuned to harness the cloud specificities. As a deployment is virtually isolated, authentication between the nodes is redundant, so it was removed. Also, the chunk size of data read/written was increased to 1 MB for a higher throughput.

BitTorrent. For data patterns like broadcast or multicast, having additional replicas enables to increase throughput and to balance the load while clients collaborate to retrieve the

data. Thus, for scenarios involving replication (above a user-defined threshold) of large datasets, we rely on BitTorrent. We use the MonoTorrent [28] library implementation of the protocol. We further built on this approach to gain performance for time critical transfers or highly accessed files by increasing the number of replicas at the expense of occupying extra storage space. Implementation-wise, the default protocol packet size of 16 KB was changed to 1 MB, which our experiments showed to increase the throughput by 5 times. Trackers are deployed in each VM, at which the peers (i.e., the Transfer Managers using the torrent module) register or query for seeder discovery.

The cost of data dissemination. As replication has a direct impact on the transfer performance, we propose a cost model that gives hints on how to adapt the replicas count with respect to transfer speedup. We start by associating a cost for storing data on a VM disk. Although there is no additional cost for the local virtual disks, the storage capacity is fixed with respect to the VM type. Thus, the cost can be define as the capacity of the VM disk over the renting cost: $cost_{MB} = \frac{Local\_Disk\_Capacity}{VM\_Pricing}$. Then, the cost of having N_R replicas, each having $Size$ MB, is $cost_{Replicas} = N_R \times Size \times cost_{MB}$. Next, we examine the time gain obtained from each replica. Based on empirical observations, we assume a linear dependence for the transfer time when the number of replicas varies between 1 and the number of nodes. On the one hand the time to transfer the file to the VM is $time_{transfer} = \frac{Size}{Throughput}$ for one replica. On the other hand, when there is a replica on all nodes ($N_R = N_{Nodes}$), than the transfer time is 0, . This leads to the next function defining the gained time:

$$time_{gain} = \frac{Size}{Throughput} * (1 - \frac{N_{Nodes} - N_R}{N_{Nodes} - 1}),$$

varying from 0 for one replica up to $time_{transfer}$, when data is already present and no transfer is needed. Thus, we are able to associate a cost for speeding-up the data dissemination by considering the ratio $\frac{time_{gain}}{cost_{Replicas}}$. Users can define a certain threshold cost, i.e., $user\_def\_cost\_ratio\_threshold$, that they are willing to pay for speeding the transfer. This cost constraint will be used by the Replication Agent to scale the replicas in the system and choose one transfer module over another in order to decrease the transfer time within the cost boundaries imposed by the extra storage space used, as shown in Algorithm 1.

Unlike for MapReduce processing, the cloud workflow ecosystem is richer. Thus we have integrated this TomusBlobs approach for workflow file management into an existing engine, the Microsoft Generic Worker workflow, by replacing its default data storage backend, which relied on Azure Blobs. The Generic Worker engine was selected, as it facilitates the process of porting existing science applications to clouds (in particular Azure) by deploying and invoking them with minimal effort and predictable cost-effective performance. While the system is optimized for Azure, it can be easily extended to other cloud technologies. The Generic Worker supports this generalization through a set of pluggable components with standardized interfaces that allow a simple migration to other clouds. One only needs to replace the runtime component with one using the specific API of other platforms. This architecture facilitated the integration as it allowed to simply plug our storage approach once implementing the provided API.

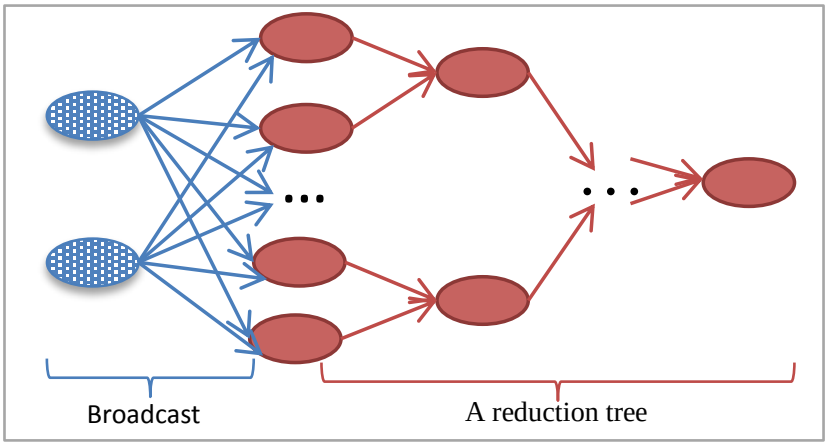


Figure 7.8: Workflow schema for the synthetic benchmark.

7.5 Validation and Experimental Evaluation for Workflows

In this section, we focus on evaluation the benefits brought by TomusBlobs in the context of workflows. As before, the experiments were performed on the Azure cloud, using 100 cores, with experiments repeated hundreds of times at various moments. The cores were distributed across 50 Medium (2 CPU cores, 3.5 GB memory, 490 GB local disk) VMs. The focus of this evaluation lays on the impact of TomusBlobs on the workflow makespan and I/O phase.

7.5.1 TomusBlobs in the Context of Synthetic Workflows

To analyze the adaptive behavior and the performance of TomusBlobs for workflows, we start with a synthetic setting. To this end, we implemented a simple benchmarking workflow (Figure 7.8) that encapsulates two data access patterns (broadcast and pipeline within a reduction tree). The workflow is composed of 38 identical jobs, with 20 of them on the first tree layer. Each job takes 2 input files containing numbers, applies an operation and stores the result in an output file, used by the tasks on the next layers. Additionally, 2 other jobs (the left ones in Figure 7.8) are used for staging-in the initial input files. This workflow is executed using the Generic Worker workflow engine using two data management backends: the default one relying on Azure Blobs, and the TomusBlobs approach for workflows discussed in Section 7.4.

Scenario 1: Small files, no replication. In the first scenario we evaluated the performance that each of the default-provided transfer methods can bring when used for handling the files of the workflow. This scenario is particularly useful to determine the threshold up to which the in-memory transfer is efficient for cloud VMs. Figure 7.9 a) displays the average time of the workflow jobs to read the 2 input and share the output file. Not surprisingly, managing the files inside the deployment reduces the transfer times up to a factor of 4, compared to the remote shared cloud storage (Azure Blobs). As expected, the in-memory solution delivers the best results for small files. When file sizes are larger, transferring directly becomes more efficient, as the in-memory module has to handle more fragments -

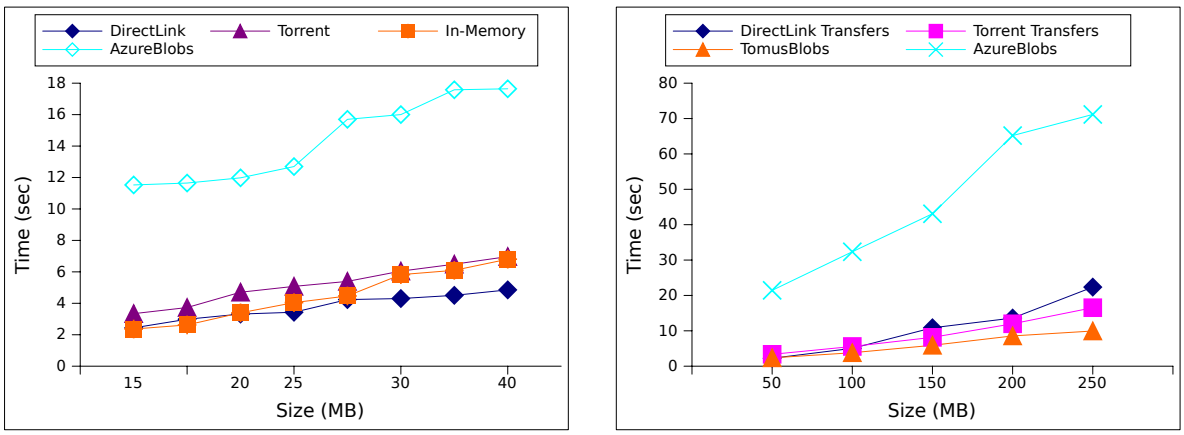


Figure 7.9: The I/O time per job, when 2 input files are downloaded and 1 is uploaded.Small (a) Left) and large (b) Right) file sizes are considered.

the files are fragmented/defragmented into 4 MB chunks (the maximum size of an object in Azure Caching).Another consequence of the fragmentation is a higher performance variation, which increases up to two times. Avoiding such variations is important for scientific applications require predictable performance [149]. Based on these observations, a threshold of 15 MB for the size of the files shared in-memory seems appropriate. Finally, the torrent module pays the price of an extra operation for hashing the file and generating the ".torrent" metadata, used by peers for download, making this strategy inefficient for small non-replicated files.

Scenario 2: Medium to large files, replication enabled. Next, we evaluate the impact of our approach and its ability to adaptively switch between transfer methods in a more complex scenario: sharing large files, replicated across the cloud infrastructure. The stage-in jobs (the ones in the left of Figure 7.8) of our synthetic workflow generate 5 replicas for their output data; in Azure Blobs the number of replicas is automatically set to 3 and the files are transparently distributed within the storage service. We notice from Figure 7.9 that the adaptive behavior of our solution which mixes the transfer strategies leads to a 2x speedup compared to a static file handling (also leveraging deployment locality) and a 5x speedup compared to Azure Blobs (remote storage). With multiple seeders (i.e., providers) for the replicas, the torrent-based module is more efficient than direct transfers. Thus, in the broadcast phase of the workflow, torrents perform better, while in the reduction phase, direct link will work better for the pipeline transfers. Finally, our adaptive solution exploits these patterns and switches in real-time between these modules in order to provide the best option each time according to the context.

7.5.2 Using TomusBlobs to Execute a Biological Workflow Application

The final set of experiments focuses on a real-life scientific workflow. We illustrate the benefits of our approach for the BLAST application, a workflow for comparing primary biological sequences to identify those that resemble. The workflow is composed of 3 types of batch jobs. A *splitter* partitions the input file (up to 800 MB in our experiments) and distributes it

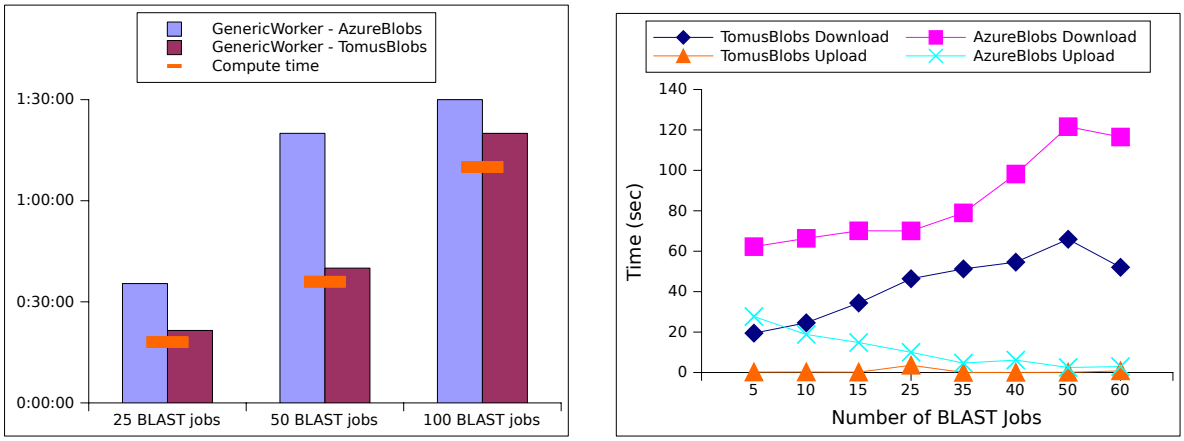


Figure 7.10: a) Left: The BLAST workflow makespan: the compute time is the same (marked by the horizontal line), the time difference between the approaches comes from the data handling. b) Right: Average times for staging data in and out for a Blast job when the number of jobs is increased.

to the set of distributed BLAST jobs. The core algorithm (the *BLAST jobs*) matches the input file with reference values stored in 3 database files (the same for all the jobs). Finally, the *assembler* job aggregates the result from the BLAST jobs into a final result.

Figure 7.10 a) presents the makespan of executing the BLAST analysis while Figure 7.10 b) reports, for the same experiment, the average file upload and download times per BLAST job. As before, the experiments were carried out with the Generic Worker using Azure BLOBs and our adaptive solution. Increasing the number of jobs results in smaller temporary files produced by the BLAST task. However, the size of the input database files to be broadcast to all jobs remains the same ($\sim$ 1.6 GB). As the number of nodes available for the experiment was fixed (50 Medium type VMs), the tasks are executed in waves when their number exceeds the VMs. This explains the drop in the average transfer time when running more than 50 BLAST jobs (Figure 7.10 b), as the few remaining jobs from the second wave will have a faster execution time as they incur less concurrency. We notice that the computation time is significantly reduced by adapting the transfer method to the data access pattern. We observe that the file handling times are reduced to half per workflow job when using the TomusBlobs approach.

7.6 Discussion

Porting data-intensive scientific applications to the clouds raises many challenges in exploiting the benefits of current and upcoming cloud infrastructures. Efficient storage and scalable parallel programming paradigms are some critical examples. To address these challenges, we introduced TomusBlobs, a cloud storage solution aggregating the virtual disks on the compute nodes, validated as storage backend for a MapReduce framework, and its extension for workflow file management. We demonstrated the benefits of our approach through experiments on hundreds of nodes using synthetic benchmarks as well as real-life appli-

cations. The evaluation shows that it is clearly possible to sustain a high data throughput in the Azure cloud thanks to our low-latency storage: TomusBlobs achieves an increase in throughput under heavy concurrency of up to 5x for writes and up to 3x for reads, compared to Azure BLOBs. In the context of workflow data management, our solution brings a transfer speed-up of up to a factor of 5x compared to using the default cloud storage and a factor of 2x over local default file management. When using TomusBlobs as a storage backend for our MapReduce framework in A-Brain, we achieved speedup times of up to 2x compared to Azure MapReduce. Finally, these benefits are complemented also by the significant cost reduction of up to 30% that our approach provides.

Let us note that the approach can be further extended with a performance model which considers the cloud's variability and provides a self-adaptive and self-optimize behavior by means of predictions. From the perspective of the A-Brain application, TomusBlobs helped to show that the univariate analysis is not sufficient for finding correlations between genes and brain images as it is to sensitive to the outliers from the data set. Hence, the robust statistical machine learning technique needs to be employed. As this analysis will greatly increases the computation workload, by up to 20 times, it is necessary to scale the processing beyond the resources which we can acquire in a single data center deployment. We address this in the next chapter by scaling the processing across multiple data centers.

Chapter 8

Going Further: Scaling MapReduce across Multiple Data Centers

Contents

8.1 Map-IterativeReduce: Handling Reduce-Intensive Workloads 77

8.2 Geographically Distributed MapReduce 81

8.3 Validation and Experimental Evaluation 83

8.4 Discussion 89

This chapter develops the contributions published in the following papers:

- *MapIterativeReduce: a framework for reduction-intensive data processing on azure clouds.* Radu Tudoran, Alexandru Costan, and Gabriel Antoniu. In Proceedings of third international workshop on MapReduce and its Applications (MapReduce 2012) held in conjunction with HPDC 2012, Delft, The Netherlands, June 2012, pp. 9 - 16
- *A performance evaluation of Azure and Nimbus clouds for scientific applications.* Radu Tudoran, Alexandru Costan, Gabriel Antoniu, and Luc Bougé. In Proceedings of the 2nd International Workshop on Cloud Computing Platforms (CloudCP 2012), held in conjunction with EuroSys 2012, Bern, Switzerland, April 2012, pp. 1-6
- *TomusBlobs: Scalable Data-intensive Processing on Azure Clouds* Alexandru Costan, Radu Tudoran, Gabriel Antoniu and Goetz Brasche. Journal of Concurrency and Computation: Practice and Experience 2013

Commercial clouds traditionally support web and small database workloads. However, with the emergence of data science, an increasing number of scientific analysis are being migrated to clouds. In practice, many of these applications, such as the statistically robust analysis required by the A-Brain application, are more resource-demanding than the typical cloud applications. Executing such demanding workloads within a cloud site could induce two undesirable situations: 1) other cloud users do not have enough resource to lease on-demand in a particular data center; 2) the computation creates performance degradation for other applications in the data center, e.g., by occupying the network bandwidth or by creating access contention to the cloud storage service. One way to avoid these situations is to divide the computation into smaller sub-workloads and to execute them within different data centers. The site locations where to execute these resource-demanding applications can be chosen in collaboration with the cloud provider.

As we move to the world of Big Data, single-site processing becomes insufficient in many scenarios. After all, one of the the founding idea of grid computing was to provide uniform data processing across multiple sites, based on the assumption that control over how resources are used stays with the site, reflecting local software and policy choices. Leaving control to individual sites was a pragmatic choice but also led to a point beyond which grid computing found it hard to scale. In contrast, clouds, which are easier to use and manage than grids, let users control remote resources, opening the path for geographically distributed computing over domains from multiple sites. Several advantages arise from running computations on multi-site configurations: higher resilience to failures, distribution across partitions (e.g., moving computation close to data or vice-versa), elastic scaling to support usage bursts, load balancing and larger computation power harvest from the available resources in each site.

Lack of support for multi-site processing. As discussed in Section 7, clouds offer limited support for processing and managing scientific data. The few processing tools which exist today, typically MapReduce-like frameworks, are built for single-site or single-cluster processing. In fact, the cloud model maps all users deployments to a single data center. Therefore, any user-deployed platform is a single-site compute engine and is limited scale-wise to the available resources within the data center. Some efforts have been made to address the challenges of distributing the computation beyond a data center. In [98], the authors implement the Elastic MapReduce API using resources other than the Amazon EC2, such as the ones from private clouds, obtaining a MapReduce framework for hybrid clouds. In [121], the MapReduce model is extended hierarchically to gather resources from multiple clusters while scheduling the jobs based on load-resource capacity or location of data. However, there are no solutions focusing on distributing the processing across deployments running over multi-site of a public cloud provider, let alone offering efficient data-management support in the context of MapReduce scenarios.

Full reduction across sites is complex. In addition to the lack of explicit support for multi-site MapReduce, current processing engines do not support full reduction of the results (e.g., in the MapReduce model, the number of final results is given by the number of Reducers). This is a major setback for computing a global result for reduce-intensive workloads. Moreover, in the case of multi-site setups, this lack of functionality can lead to many expensive (time and money wise) inter-site data transfers of the reducers results in the process setup

to assemble the global result. One way to obtain this global result is for programmers to implement an additional "aggregator" that collects and combines the output data produced by each reduce job. For workloads with a large number of reducers and large data volumes, this approach can prove inefficient as it performs the computation sequentially on a single entity.

Some recent efforts, detailed in Chapter 5, introduce support for *iterative computation* into the MapReduce engine, allowing to apply the same operator successively on the data. HaLoop [32] was built on top of Hadoop to support iterative computation, relying on a loop-aware task scheduler and on loop-invariant data caching. Twister [53] and its Azure declination, Twister4Azure [78], employs a light-weight MapReduce runtime system and uses publish/subscribe messaging-based communication instead of a distributed file system and data caching. iMapReduce [185] provides support for common features of iterative algorithms, eliminating the shuffling phase and maintaining persistent tasks and persistent socket connections between tasks. All of these frameworks target applications with iterations across MapReduce workloads but they could hardly be used to efficiently tackle the problem of globally reducing the final results to a unique one: multiple MapReduce iterations with identity map phases would have to be created. This leads to an extra overhead due to loop control mechanisms and to job scheduling across iterations. In contrast, message-passing runtime systems such as MPI provide support for reduction through a rich set of communication and synchronization constructs. However, they suffer from little fault tolerance support, which impacts the applications' perceived reliability on clouds, mainly built on commodity hardware.

Our approach in a nutshell. To address these challenges of executing Big Data applications across multiple sites, we propose a two-tier hierarchical MapReduce scheme. The bottom tier distributes TomusBlobs-MapReduce instances across cloud data centers. The top tier computes the global final result. The input data and the processing jobs are split among all the available resources, harvesting the available compute power from multiple cloud sites, while the global result is aggregated using a Meta-Reducer. To minimize the data exchanges between data centers and address reduce-intensive workloads, we propose a Map-IterativeReduce technique that efficiently schedules the reduce process in parallel, based on a reduction tree, to compute a unique result. Additionally, Map and Reduce jobs can be interleaved as we eliminate the usual barrier between these two phases. Using these approach for a multi-site MapReduce, we are able to achieve high scalability in the clouds reaching 1000 cores across 4 deployments and 3 data centers. Performance-wise, we are able to reduce the data management time by up to 60% compared with cloud-based solutions.

8.1 Map-IterativeReduce: Handling Reduce-Intensive Workloads

In this section we introduce Map-IterativeReduce, a technique for efficiently computing a unique output from the results produced by MapReduce computations. Targeting mainly the reduce-intensive workloads, our approach provides support for a set of model extensions which favor parallel efficient scheduling of the Reduce operation jobs in order to combine all results to a single one. At the same time, it preserves the cleanness of the MapReduce programming paradigm, while facilitating its use by a large set of applications. Our solu-

tion was validated on the Azure cloud and leverages TomusBlobs (presented in Section 7) for low-latency and high-throughput under heavy concurrency data accesses. Using this approach, we are able to reduce within each site the results of the MapReduce computation and thus, to minimize the number of transfers across sites when aggregating the global result in the multi-site hierarchical MapReduce.

We introduce a set of core design principles in order to extend the MapReduce model to support an iterative reduction scheme.

- In Map-IterativeReduce, *no synchronization barriers* are needed between the Map and the Reduce phases, the reducers starting the computation as soon as some data is available. Thus, by not waiting anymore for the completion of the slowest or latest mappers the total execution time is reduced.

- Our approach builds on the observation that a large class of scientific applications require in fact *the same Reduce operator* from the MapReduce process to be applied also for combining the data to a single output. In order to exploit any inherent parallelism in the reduction phase this operator has to be at least *associative and/or commutative*. In fact, most reduce operators that are used in scientific computing for combining the results (e.g., max, min, sum, select, filter etc.) are both associative and commutative. Reduction may be also used with non-associative and non-commutative operations but offers less potential parallelism.

- Considering that results are accumulated down the reduction tree, there is *no need for any (centralized) entity* to control the iteration process, check the termination condition or collect data from reducers, as in vanilla or iterative MapReduce implementations.

We present the data flow in a Map-IterativeReduce computation on Figure 8.1. The model consists of a classical *map* phase followed by an *iterative reduce* phase. A reducer applies the associative reduction operator to a subset of intermediate data produced by mappers or reducers from previous iterations. The result is fed back to other reducers as intermediate data. The processing starts as soon as data becomes available from some (but not necessarily all) mappers or reducers. We recall that in the default MapReduce implementations, such as the one offered by Hadoop, this is not the case as there is a synchronization barrier between the map and reduce phases. However, we observed that this makes the performance of typical single-step MapReduce to go down sharply, when the availability of these intermediate results is subject to different latencies. Hence, we opted for *eliminating the input barrier* of reducers, leveraging fully the commutativity assumption about the reduce operations. The successive iterations of the reduce phase continue thus asynchronously, until all the input data is combined. At the end of each iteration, all reducers check whether their output is the final result or just an intermediate one, using several parameters attached to the reduce tasks. This avoids the single point of failure represented by a centralized control entity. Such centralized components are typically used in the existing iterative MapReduce frameworks for evaluating the termination or convergence condition and in turn to schedule the jobs of the next iterations.

With this model, we formally define the reduction as a scheduling problem: we map reduction tasks to a pool of reducers using a *reduction tree*. Indeed, the iterative reduce phase can be represented as a reduction tree, each iteration corresponding to a tree level. For a

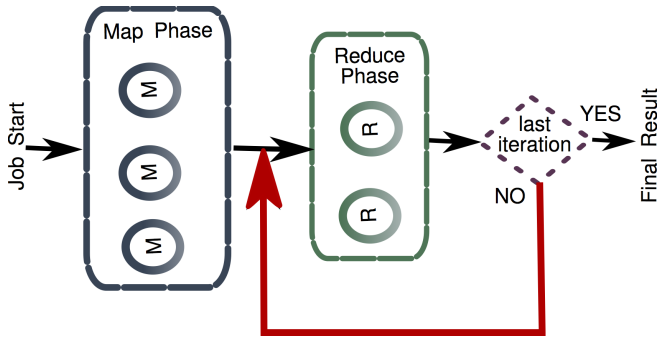


Figure 8.1: The Map-IterativeReduce conceptual model

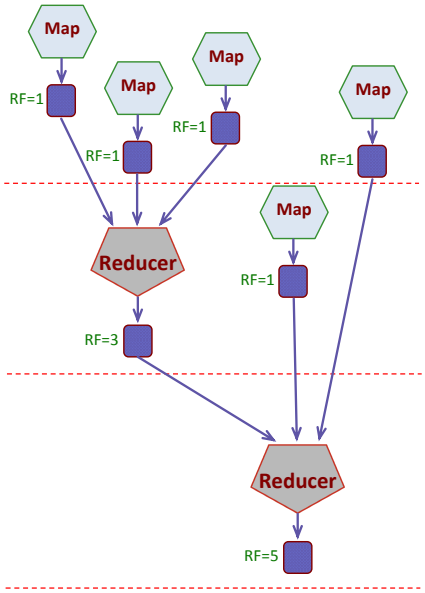


Figure 8.2: An example of a reduction tree for 5 mappers and a reduction ratio of 3

better understanding, consider the simple example of an unfolding of the iterative process into a reduction tree depicted on Figure 8.2. It presents a scenario in which each reducer processes 3 sets of data at a time, initially issued from 5 mappers. In the first iteration, only one reducer has data available; in the second one, the bottom reducer produces the final result by combining the remaining intermediate data from the slowest mappers with the intermediate data produced by the first reducer. With the associative and commutative operators, the computations corresponding to different iterations can interleave and exploit the inherent parallelism of the Reduce tasks. In this case, the second reducer can start processing data from the last two mappers in parallel with the other reducer. We introduce two parameters that control the scheduling of the iterative process and the unsupervised termination condition, i.e., the termination condition is not supervised and checked by a centralized component.

The Reduction Ratio defines the workload executed by a reducer, i.e., the number of intermediate results to be reduced within the job. This parameter together with the number of map tasks completely determine the number of reduce jobs that are needed in the iterative process and the depth of the unfolded reduction tree. In the previous example, the Reduction Ratio was 3, and in conjunction with the number of Map jobs (5) determines the number of iterations (2) and the number of reducers that need to be used (2). Unlike the collective operations which only use binary operators, as in MPI, we are able to support n-reductions, which is more generic and can be adjust to particular loads. Having this feature is important because most often, the optimal reduction ratio is greater than two, as illustrated bellow.

The Reduce Factor defines the termination condition, checked locally by each reducer. Since the goal was to process all data into a single result, the termination condition

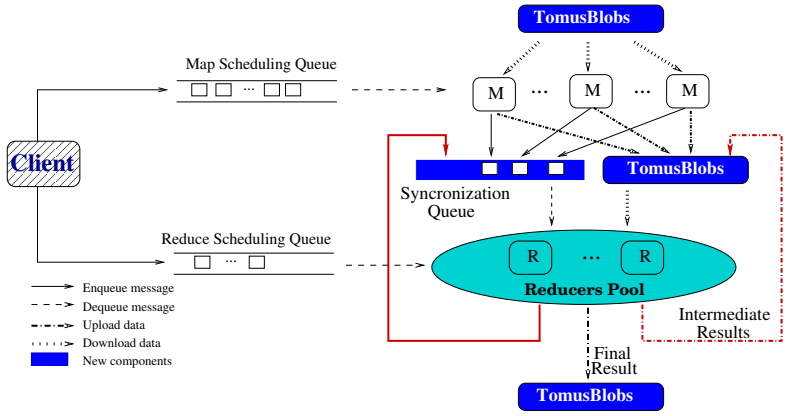


Figure 8.3: The architecture of the Map-IterativeReduce framework

is answering the question *are all the results reduced?*. The Reduce Factor (RF) is used for measuring the progress towards this goal. The initial intermediate data, coming from mappers, will have a Reduce Factor of 1, while the intermediate results produced by reducers will have a Reduce Factor equal to the sum of the factors of all inputs. Therefore, the termination condition becomes a comparison between the resulting factor and the total number of jobs. In the previous example, the top reducer has a resulting Reduce Factor of 3 while the last one will have a factor of 5, equal to the number of mappers and therefore marking the termination condition.

We extended the TomusBlobs–MapReduce framework (Section 7) with this iterative reduce approach in order to validate its benefits for scientific applications on the one hand, and to enable efficient multi-site MapReduce on the other hand. The architecture is presented in Figure 8.3. It relies on the same loosely coupled components as before, only extending their functionality. The *Client* has two roles. First, it is in charge of submitting the workload (i.e., map and reduce jobs). Second, it determines the number of reduce jobs in the iterative process which is computed based on the the number of mappers and the Reduction Ratio selected by the user. The *Mapper* functionality remains the default one. The *Reducer* is the component that implements the iterative reduction process in two steps. In a first phase, it receives the workload description from the Client (including the *Reduction Ratio*). Then, it starts to process the intermediate results updating the *Reduce Factor* of the resulting tasks. After each iteration, the termination condition is checked. If the *Reduce Factor* of the computed data equals the number of mappers in the workload, the iterative process ends. Otherwise, the Reducer will behave like a Mapper and will prepare its result as an intermediate data for the Reducers in the next iterations. Communication-wise, the same mechanisms are used as before: Azure Queues are used as a scheduling mechanism for the jobs and TomusBlobs is used as a storage backend for leveraging data locality and for enabling high throughput under heavy concurrency.

One of the key reasons for the success of MapReduce frameworks is the runtime support for fault tolerance. Extending TomusBlobs-MapReduce to support iterative reduction does not weaken this property. Our approach for dependability is two-folded: on the one hand, we rely on the implicit fault-tolerance support of the underlying platforms (Azure PaaS, TomusBlobs); on the other hand, we implemented specific techniques for dealing with failures in Map-IterativeReduce. For the availability of data, we rely on the replication mechanisms of TomusBlobs. Regarding the jobs to be executed, we use the visibility timeout of the Azure

Queues to guarantee that a submitted message describing the jobs will not be lost and will be eventually executed by a worker, as in TomusBlobs-MapReduce. For the iterative reduce phase, however, this mechanism is not sufficient. Since a Reducer has to process several messages from different queues, a more complex technique for recovering the jobs in case of failures is needed. Thus, we developed a watchdog mechanism distributed in each node running a reducer. Its role is to monitor and to log the progress of the Reducers. For this, it implements a light checkpointing scheme as it persistently saves the state of the Reduce processing (i.e., it logs the intermediate inputs that were processed by the reducer). In case of a failure of some reducer, the reduce job will reaper in the scheduling queue and will be assigned to another reducer from the pool. Next, the watchdog monitor system of the reducer now in charge of the job will check for existing log information corresponding to the reduce job. If any, it checks whether the watchdog from the previous reducer is still active, in order to avoid duplication due to slow executions. If it is not the case, the watchdog, will rollback the descriptions of the intermediate data processed previously up to the point of failure. This allows Map-IterativeReduce to restart a failed task from the previous iteration instead of starting the reduction process from the beginning.

8.2 Geographically Distributed MapReduce

Handling Big Data scientific applications on the clouds requires many compute resources which are not always available in a single data center. We address this challenge through a divide-and-conquer strategy: the workload is split into smaller sub-problems which will be executed in different deployments across multiple data centers. Therefore, we propose a multi-site hierarchical MapReduce scheme. Several building blocks underpin the creation of this solution that is able to harvest the available resources within each cloud site. This data processing scheme in which data is partitioned, scattered and processed in parallel, relies on a layered data access model built on top of different storage systems in order to support both storage and transfers. At an abstract level, the system needs to provide an end-user environment for the multi-site hierarchical MapReduce that provides a uniform and transparent abstraction independent of any particular cloud deployment and that can be instantiated dynamically. We examine in this section the mechanisms to accomplish this.

The goal of the multi-site hierarchical MapReduce is to setup a geographically distributed compute platform, large enough to support execution of Big Data applications, and able to handle data across this virtual setup. The conceptual architecture is presented on Figure 8.4. Rather than simply selecting the site with the largest amount of available resources, we select locations from a few different domains and build the environment on top of them. The hierarchical MapReduce consists of two tiers.

- At the bottom level, distinct instances of TomusBlobs-MapReduce are deployed at each site. The Map-IterativeReduce processing scheme is enabled within each such instance in order to reduce locally the number of MapReduce outputs and in this way to minimize the inter-site data exchanges.

- In the top tier, the aggregation of the global result is computed with the help of a new entity that we introduce, called *MetaReducer*. Its role is to implement a final reduce step to aggregate the results from all sites as soon as they become available.

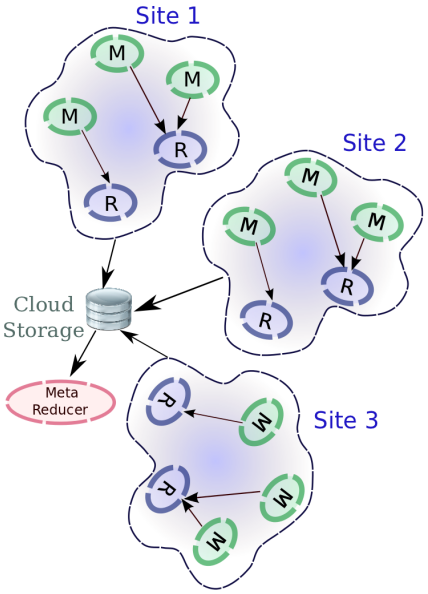


Figure 8.4: The scheme for a MapReduce computation across multiple sites

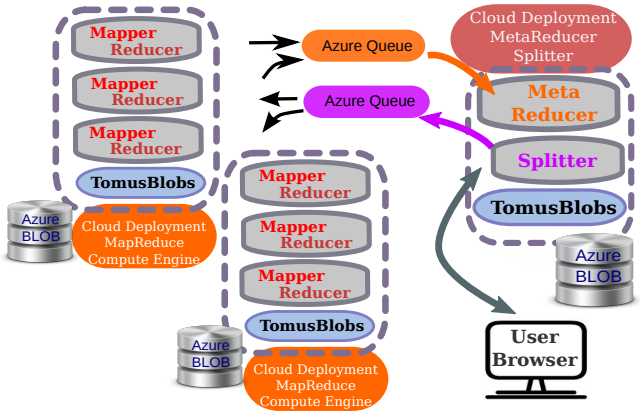


Figure 8.5: The architecture for the multi-site MapReduce

Data manipulation is a key aspect to enable such a geographically distributed processing across cloud data centers. We use several storage systems to optimize data management within the same deployment or between deployments. Analyzing the data processing phases, we have identified three main scenarios that need to be addressed with the proper data management solution.

Input/Output Data is handled using the *Cloud Storage* service instance within each site. Despite being located in the same data center, the storage is remote with respect to the computation and incurs high latencies and usage costs. In our system, it is mainly used to transfer the initial data, typically coming from on-premises, to be processed inside the deployment.

MapReduce Data (Intra-site) is handled via the *TomusBlobs* approach in order to enable fast data access within the deployment. The choice for this storage solution for this data processing phase is supported by its abilities to provide high-throughput under the heavy concurrency introduced by mappers. Furthermore, it reduces the costs of handling the data, while federating the local virtual disks.

Inter-Site Data Exchanges is performed using the *Cloud Storage* service instance that is located on the site where the Meta-Reducer is deployed. Selecting this storage option was straightforward: since the deployments are independent and virtually isolated, they must communicate through a storage repository that is accessible from outside the deployment. These transfers are quite expensive in terms of costs and latency. Thus, the goal is to minimize them as much as possible, relying in this sense on the Map-IterativeReduce technique that we devised.

We implemented the multi-site hierarchical MapReduce approach on top of the Azure clouds. The architecture is presented in Figure 8.5. We added support for communication between the TomusBlobs-MapReduce and the Meta-Reducer and designed a scheduling mechanism for scattering (i.e., partitioning) the initial data across multiple sites. The Meta-Reducer was implemented as an independent service built on top of a pool of reducers. The number of communication queues used, in our case Azure queues, is proportional with the number of deployments. Additionally, we used a strategy of hashing data inside the same deployment, if the same data is replicated across multiple storage centers, in order to minimize data transfers. We introduce a new entity, called *Splitter*, for partitioning the workload between the MapReduce processes. The Splitter has the role to segment data between sites such that each deployment will work on a sub-problem. Additionally, the Splitter is in charge of creating the corresponding job descriptions (i.e., Map and Reduce tasks) for the MapReduce engines and sending them to be processed via the queueing communication mechanism of TomusBlobs-MapReduce. As a final optimization, data can be replicated across sites or pre-partitioned within the data centers in order to reduce the cost of staging-in the initial data to each site.

8.3 Validation and Experimental Evaluation

In this section, we assess the benefits that the proposed Map-IterativeReduce technique can bring to scientific applications and for multi-site MapReduce processing. Moreover, we test the performance gains and the computation scale that can be achieved when running MapReduce processing across multiple cloud data centers. The final goal is to understand what are the best options that have to be put in place for running a large-scale, long-running scientific simulation, as the one needed for the A-Brain analysis. To this extent, the experiments presented here scale up to *1000 cores*, a premiere for scientific applications running on Azure.

The experiments were performed in the Microsoft Azure cloud, with deployments across West Europe, North Europe and North-Central US data centers. The number of nodes within a single deployment varies between 200 cores, used in the analysis of the iterative reduce technique, up to 330 cores when evaluating the multi-site MapReduce. In terms of number of distributed deployments running hierarchical MapReduce we have scaled up to 4 deployments accumulating 1000 cores from 2 data centers. One of the objectives of this section is to evaluate different options for running large-scale experiments. To this purpose, we have analyzed the performances obtained when using different VM types, considering Small instances (1 CPU core, 1.75 GB of memory and 225 GB of disk space) as well as Medium or Large VMs which offer 2 times, respectively 4 times, more resources than the Small ones. The BlobSeer backend of TomusBlobs was used with the following configuration: 2 nodes for the version and provider managers, 20 nodes for the metadata servers and the rest of the nodes were used as storage providers.

8.3.1 Selecting the VM Type: Impact of Multi-Tenancy on Performance

Cloud providers work on several mechanisms to virtually isolate deployments with the goal of offering fairness to users despite the multi-tenant environment. In this context, we refer

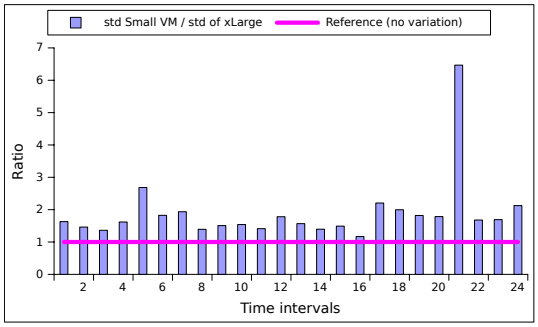


Figure 8.6: The variation of standard deviation in the context of multi-tenancy

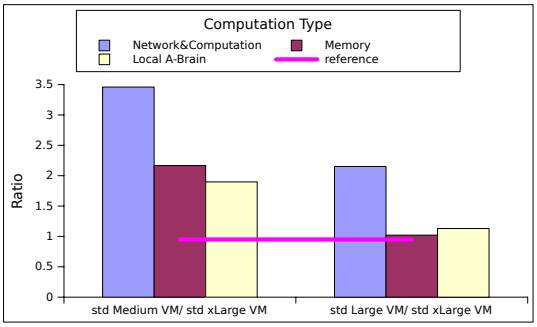


Figure 8.7: The impact of multi-tenancy on different applications: Network & Computation Intensive; Memory Intensive Computation; Local A-Brain.

to fairness as providing the performance that users are paying for. The question of how multi-tenancy affects performance remains an open issue, and this is what we explore in this subsection.

As all other commercial clouds, Azure offers different types of VMs. The biggest one, *ExtraLarge* also referred to as xLarge, fully occupies the physical machine. For the other VM types, the physical machine is shared with other VMs. However, the virtual cores rented are map to physical CPU, i.e., there is a one to one relationship between VMs virtual cores and physical cores. Therefor, the non-xLarge VMs are collocated on the physical machine in the limit of the available cores (usually 8). Even if there are no interferences at CPU level, a degradation of performance can be expected when accessing the local (shared) storage, memory or network. We propose the following evaluation methodology: we measure the variability that application have for different VM types and compare it with the one obtained with the xLarge VM. This is considered as a reference of the stable performance because this type of instances is not subject to multi-tenancy on the physical node, everything being isolated except the network.

In a first experiment we considered a declination of the A-Brain analysis, where a small computation subset is executed locally i.e., data is read from the local disk, the computation is performed as matrix operations and the result is stored back to the local disk. We repeated this experiment several days for 1440 times on each xLarge and Small machines. The results were grouped in pairs of 60 and the standard deviation of the time period was computed. The goal of this analysis is to determine the fairness of sharing the physical nodes among VMs belonging to different users.

Figure 8.6 depicts the analysis of the results by reporting the ratio of the two VM types variability. A value of this ratio close to 1 (the reference line) in Figure 8.6 would indicate that either the neighboring applications do not affect the performance, or that there are no neighboring applications. Such a point can be seen for the 16th time interval. A high value for the ratio would show that the performance of the instance is affected by external factors, as is the case for the 5th and 21th interval. The reason for observing both small and big values is that the samples were done in a time span of approximately 1 week, one minute apart, a time span that proved to be sufficiently large to capture various situations. Thus, the experiment demonstrates that indeed, the interferences impact on instances performances

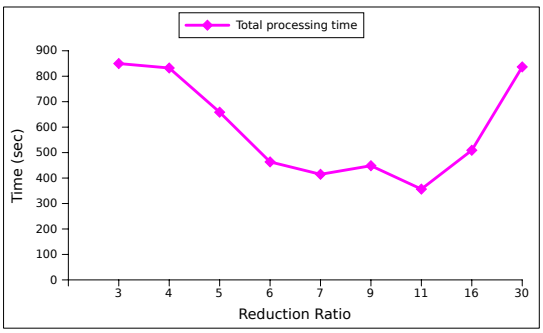


Figure 8.8: The impact of varying the *ReductionRatio* on the application execution time

Mappers	Ratio	MapReduce with Aggregator		Map Iterative Reduce	
		Reducers	Total Jobs	Reducers	Total Jobs
50	5	10	61	13	63
100	5	20	121	25	125
200	5	40	241	50	250
300	5	60	361	74	374
400	5	80	481	100	500
500	5	100	601	125	625

Figure 8.9: The experimental setting for the iterative reduce analysis

caused by the local disk I/O or memory accesses, is possible. Finally, the analysis reveals that Small instances tend to have quite an unstable and unpredictable performance and therefore are not the suitable for long-running scientific experiments on the cloud.

In the next experiment we extend the analysis by considering 3 different compute scenarios to be executed on different VM types. The first one performs computations and many network transfers, the second one performs only in-memory operations while the third one is the aforementioned local A-Brain declination. We have executed these applications on Medium, Large and ExtraLarge VMs, since the previous experiment conclusively showed the instability of the Small instances. Using the described methodology, we depict in Figure 8.7 the variability ratio for characterizing the performance fairness, i.e., we report the standard deviation of Medium and Large VMs against the one of xLarge VMs. The highest variation occurs when the (shared) network is accessed. This is expected and can be explained by the fact that the network card becomes the bottleneck more often than the physical memory or the local physical disk, and also because the connecting links between cloud nodes are not virtualized nor allocated per user. On the other hand, the lowest standard deviation is observed for the in-memory computation, which is an encouraging discovery for compute- and memory-intensive scenarios migrated to the clouds. Regarding the variability of the VM types, one can observe that Large VMs vary less then Medium ones and tend to offer as stable performances as when the entire physical node is leased. This qualifies them as a good option for executing scientific applications in the cloud with reasonably predictable performance.

8.3.2 Performance Gains with Map-IterativeReduce

Impact of the reduction tree depth on performance. We start the evaluation of the Map-IterativeReduce technique by analyzing the impact that the shape of the reduction tree has on performance. Orchestrating the reduction tree is equivalent to determine the amount of intermediate data (i.e., the *Reduction Ratio*) that a reducer should process, which in fact depends on many factors. One influencing factor is the trade-off between the time spent by a Reducer for processing data and the time needed to transfer it. Another factor is the number of Map jobs as it determines the number of leaves in the reduction tree. However, even for an equal number of such jobs, two different applications, with different computation patterns

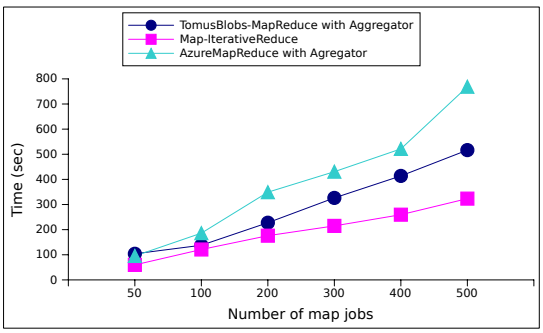


Figure 8.10: The execution times for Most Frequent Words when scaling up the workload.

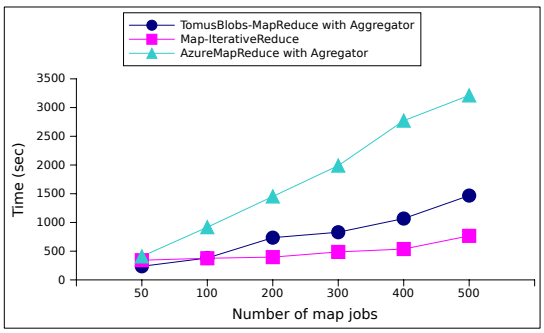


Figure 8.11: The execution times for A-Brain when scaling up the workload.

and data sizes, do not necessarily have the same optimal *Reduction Ratio*. Therefore, we evaluate the timespan achieved by different configurations for reducing (with the minimum operator) 7.5 GB of data issued from 30 mappers.

We report the results on Figure 8.8, by representing the execution time while varying the *Reduction Ratio*. We notice that the time decreases with the increase of the *ReductionRatio* up to a point where the workload scheduled for a reducer becomes much higher then the cost of transferring data and the performance drops. An analytic determination of the optimal ratio would require a performance model of the underlying infrastructure in terms of computation power, available bandwidth and would be specific for each application. However, the tree depth will always be determined by log function of the the number of leaves (i.e., $NrOfMappers$). Based on our evaluation, we can empirically determine that, in general, a reduction tree with a depth of $\sim \log_{10} NrOfMappers$ gives the best results. In the case of the experiment presented in Figure 8.8, the corresponding tree depth is 1.47. Moreover, using the approximation for the tree depth we can estimate also the optimal *Reduction Ratio*, using Equation 8.1, which applied to our experiment gives a *Reduction Ratio* of 10.11.

$$ReductionRatio \approx \sqrt[depth]{NrOfMappers} \tag{8.1}$$

Impact of Map-IterativeReduce on the performance of MapReduce scenarios. Next, we evaluate the performance gains brought by the proposed Map-IterativeReduce technique in the context of a typical MapReduce scenario: "Most Frequent Words" — derived from the widely used "Word Count" benchmark, in which the reduction was extended with a merge sort that combines and orders the occurrences of words. Thus, the processing will output as a final result only the n most frequent words. This scenario is encountered in many algorithms, such as PageRank, or scientific applications that perform statistical analysis (e.g., bio-informatics, medical studies, etc.), when users are only interested in a subset of the aggregated results, that meet some specific criteria. The experimental methodology consists in scaling the amount of data to be processed as well as the number of jobs, keeping a constant input data set (the text to be parsed) of 64 MB per map job. Therefore, as we increase the input set from 3.2 GB to 32 GB, the number of Map jobs will be proportionally scaled from 50 to 500. For this experimental setting we use a constant *Reduction Ratio* of 5 which

produces the MapReduce configuration setting presented in Table 8.9. We chose to use a constant workload per Reduce job in order to understand better the performance gains over a classical MapReduce process in which the unique final result would be obtained using an "Aggregator" entity.

Figure 8.10 shows the execution time of the "Most Frequent Words" benchmark. We compared the performance of the Map-IterativeReduce approach with two standard MapReduce engines, TomusBlobs-MapReduce and AzureMapReduce. To obtain the same computation load, each MapReduce engine was augmented with a final "Aggregator" entity that reduces the final results to a unique one. We notice that for small workloads, the Map-IterativeReduce technique and the classical MapReduce have similar performance. This is due to two reasons. First, the depth of the unfolded reduction tree is small which is almost equivalent with a set of reducers followed by the "Aggregator". Second, the latency of the initial transfers between mappers and reducers is predominant in the total execution time making any other optimizations less noticeable. In contrast, when the workload increases and the computation becomes more Reduce-intensive, the proposed approach outperforms the others, decreasing the computation time up to 40% compared with the same TomusBlobs-MapReduce framework (i.e., the same engine without the Map-IterativeReduce technique) and 60% compared to AzureMapReduce.

We continued the evaluation by studying the performance impact that the iterative reduce technique can bring for scientific applications such as A-Brain. Similarly to our previous experiment, the number of map jobs was increased from 50 to 500, while keeping the reduction ratio constant and the same configuration setup shown in Table 5.4. From the point of view of the analysis, increasing the number of Map jobs does not increase the degree of parallelism for the Map phase, but rather increases the workload as we process and test more shuffles (i.e., we increase the statistical precision of the results as detailed in Section 6). This leads to generating more intermediate results that must be processed in the Reduce phase, the overall amount of data being increased from 5 to 50 GB.

Figure 8.11 depicts the total execution time of A-Brain using the three solutions as before. We notice that for AzureMapReduce the timespan grows linearly with the workload, while the Map-IterativeReduce decreases the completion time by up to 75%. Part of this performance difference appears also between AzureMapReduce and TomusBlobs-MapReduce and is obtained due to the capabilities of the storage backends. The data proximity brought by the TomusBlobs approach significantly reduces the completion time, especially for a larger number of mappers/reducers, which can read and write in parallel from the virtual disks. The rest of the performance gain which accounts for the gap between Map-IterativeReduce and TomusBlobs-MapReduce, results due to the use of the iterative reduce method. This proves that parallelizing the reduction phase in successive iterations makes a better use of the existing resources and that it is able to reduce the computation time by up to 50%.

8.3.3 Hierarchical Multi-Site MapReduce

The compute variability of multi-site processing. We start by analyzing the performance variations that can be expected in such a distributed setup. Figure 8.13 depicts the average Map times and the corresponding standard deviation in a scenario where we progressively increase the number of jobs and the A-Brain workload. For this experiment, the initial data was split and scattered across the Azure sites where the computation was performed. Al-

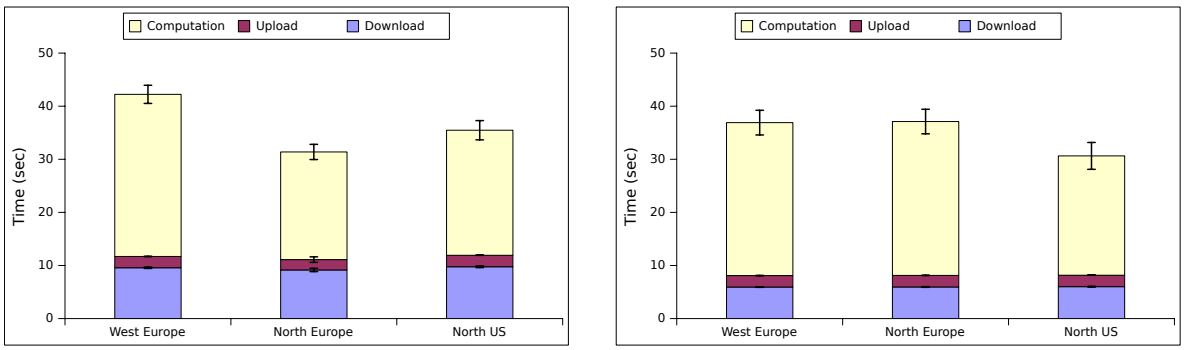


Figure 8.12: Structure of the reducers processing phases (left - reduction ratio 5 ; right - reduction ratio 3)

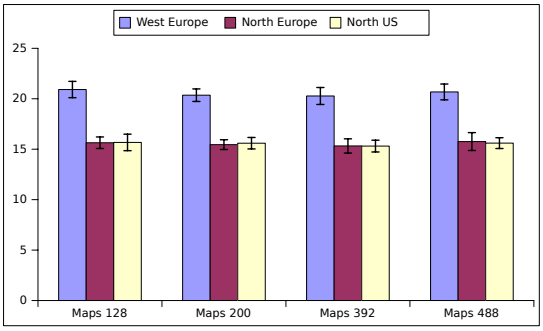


Figure 8.13: Variation of the average Map time on different sites

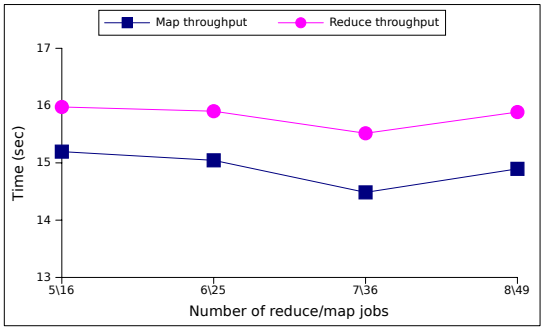


Figure 8.14: The throughput towards the intra-site storage backend for mappers and reducers. The scale shows the number of reducers and the number of mappers (i.e., reducers/mappers)

though we increase the number of parallel map jobs, the average job execution time remains approximately the same. Each mapper processes a constant amount of 100 MB of data, as in the remaining experiments presented in this section. The significance of this result is that the job time is solely determined by the data set to be processed and remains constant per input, regardless the number of jobs that are submitted to the system. In turn, this allows us complete freedom in partitioning the workload and obtaining any degree of parallelism in terms of compute jobs. Finally, it shows that TomusBlobs-MapReduce can scale and cope with large workloads even when distributing the computation across several sites. We note that the jobs executed in West Europe have slightly larger timespans than the others: this is an expected behavior since, within this analysis, the significant correlation links that were found were mostly located in the first partition of the data (assigned to this data center), generating a higher workload per task.

Depicting the reduce phase. Next, we examine the performance of the reducers and their stability. We use the same experimental setup as before and report in Figure 8.12 the execution times of the reduce jobs, divided into their components. In order to better understand

the behavior of the Map-IterativeReduce computation in such a geographically distributed setting, we vary the number of intermediate results by changing the *Reduction Ratio* parameter (shown in the left and right subfigures of Figure 8.12). It can be noticed that the computation structure in the reduce phase remains almost the same regardless the number of intermediate results to be processed. The result emphasizes the strength of our iterative reduce technique as well as it demonstrates the capabilities of our multi-site hierarchical MapReduce to support even reduce-intensive workloads. In turn, having such a steady behavior of the compute framework is important both for the application but also for the storage access time. In addition to reliability, the performance is also predictable, a feature often speculated in high-performance computing and Big Data analysis [50, 57].

The intermediate I/O phase. The final step in the evaluation consists in examining the performances that the mappers and the reducers have for managing data. We show on Figure 8.14 the average throughput that mappers and reducers achieve when accessing the TomusBlobs storage within the deployment, i.e., data access within the site. The evaluation considers an increasing workload generated by scaling the number of jobs proportionally. As before, each job processes a constant amount of data. Having more jobs translates to larger amounts of data to be processed, which in turn will produce more intermediate results. In this experiment the data size is increased from 3 GB up to 15 GB. The results of this experiment show that even in the context of multi-site hierarchical MapReduce, TomusBlobs is able to provide a steady throughput. This is achieved due to its decentralized architecture, large-scale capabilities and data-locality property it provides.

Scaling the compute workload. With all aspects of the processing performance being evaluated, we can now analyze the overall efficiency of the system and how efficiently the large workloads are being accommodated across sites, on large number of resources (i.e., 1000 cores). We used the same evaluation strategy as before: we increased the workload by generating more processing tasks which in turn produce more intermediate data. This is presented on Figure 8.15 by the increasing number of Map and Reduce jobs displayed on the abscissa. We notice a good behavior of TomusBlobs-based multi-site hierarchical MapReduce at large-scale: A-Brain's execution time increases only 3 times while the workload is increased 5 times. This is an encouraging result which demonstrates that the system can support Big Data scientific applications and that it can be used to run the statistically robust A-Brain analysis for searching through all brain–gene correlations.

8.4 Discussion

Running Big Data applications on the clouds creates new challenges for scaling the computation beyond the available resources of a data center. We have proposed a multi-site hierarchical MapReduce engine based on TomusBlobs, which enables applications to be efficiently executed using resources from multiple cloud sites. Using this approach, we were able to scale the computation in the Microsoft Azure cloud up to 1000 cores from 3 different sites. However, supporting efficient geographically distributed computation in the clouds requires careful considerations for data management. To this end, we have proposed an extension for the MapReduce paradigm which enables us to combine the output results to a

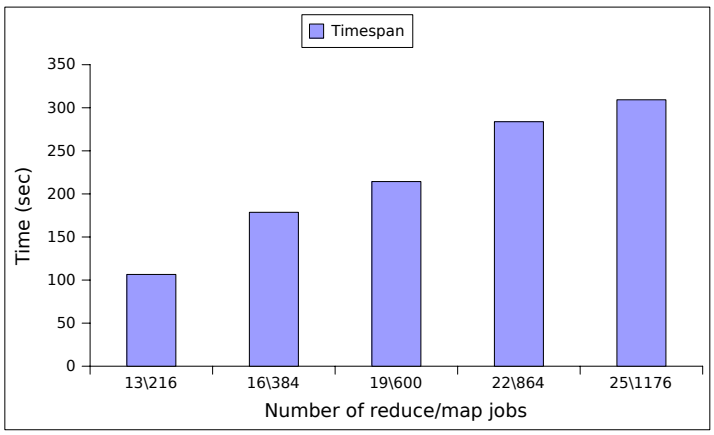


Figure 8.15: A-Brain execution time on a multi site deployment of MapReduce engines

single one, while processing them in parallel. Using this Iterative-Reduce technique we are able to minimize the number of transfers across sites and to decrease the computation time of aggregating the final result by more than half. In a subsequent step, addressed in Part III of the thesis, the hierarchical data access model can be improved in order not only to minimize the number of inter-site data exchanges, but also to improve the transfer performance. From the point of view of executing scientific applications, the system is ready to accommodate long-running experiments now that we have demonstrated its good scaling properties and its performance reliability. Therefore, in the next chapter we present and discuss the results and the lessons we learned when executing the complete A-Brain analysis on thousand of cores in a 2-week experiment.

Chapter 9

Lessons Learned : Large-Scale Big Data Experiments on the Cloud

Contents

9.1 A Large-Scale Experiment for Fitting Genotypes with Subcortical Brain Regions 92
9.2 Focusing on Long-Running Scientific Experiments 94
9.3 Addressing Data Management Issues across Data Centers 97
9.4 Discussion 102

This chapter develops the contributions published in the following paper and book chapter:

- *Machine learning patterns for neuroimaging-genetic studies in the cloud* Benoit Da Mota, Radu Tudoran, Alexandru Costan et al. In Frontiers in Neuroinformatics 2014
- *Big Data Storage and Processing on Azure Clouds: Experiments at Scale and Lessons Learned*. Radu Tudoran, Alexandru Costan and Gabriel Antoniu.In the book *Cloud Computing for Data-Intensive Applications*, to be published by Springer, 2015. Editors Xiaolin Li and Judy Qiu

As the Data Deluge is becoming a reality, the analysis of large data sets drives the scientific discovery. This process, known as data science, depends on the efficiency on the underlying frameworks which perform the processing of data. To this end, we introduced several data management techniques, described in Sections 7 and 8, to run such scientific applications at large-scale on clouds. In this section, we apply these techniques in the context of the

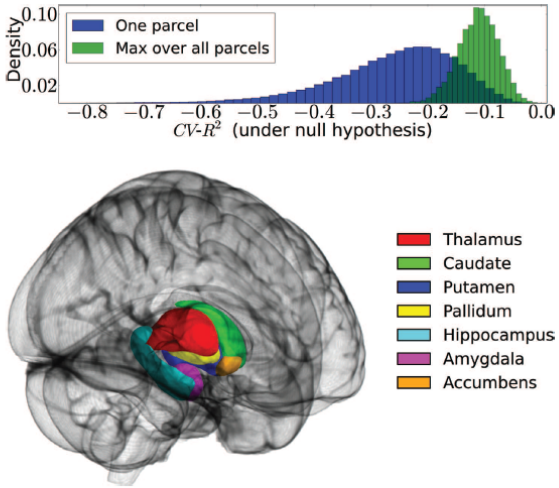
A-Brain application, presented in Section 6. The collaborative goal with the bio-informatics team is to crunch high-dimensional neuro-imaging and genetic data in order to find significant correlations between them. The motivation for this pioneer study is to enable joint analysis between genes and neuro-images, with implications in understanding the variability between individuals and brain pathologies risk factors. The analysis was performed on 1000 cores running for 2 weeks in 4 deployments across Azure data centers. As a result, it was shown for the first time how the functional signal in subcortical brain regions can be significantly fit with genome-wide genotypes. This study demonstrates the scalability and reliability of our solutions for long-running scientific applications. In addition to the biological result, performing such a Big Data analysis taught us important lessons about the issues that need to be addressed next.

9.1 A Large-Scale Experiment for Fitting Genotypes with Subcortical Brain Regions

In this section we present how the techniques introduced in Section 7 and 8 are used to execute the A-Brain analysis, described in Section 6. We recall that this is a joint effort with the Inria Saclay Parietal Team, in the context of the Microsoft Research - Inria Joint Center, for enabling large-scale analysis on clouds for bio-informatics. The A-Brian study considers functional MRI (Magnetic Resolution Imaging) contrast, corresponding to events where subjects make motor response errors. A number of 1459 subjects remained for the final analysis after discarding the ones with to many missing voxels (i.e., volumetric pixels elements which here apply to the first input data set of brain images) or with bad task performance. Regarding genetic variants, 477,215 SNPs (Single Nucleotide Polymorphism, which apply to the second input data set of genetic material) were available. Age, sex, handedness and acquisition center were included in the analysis as a third data set to account for. The analysis considers the functional signal of 14 regions of interest (ROI), 7 in each hemisphere: thalamus, caudate, putamen, pallidum, hippocampus, amygdala and accumbens (see Figure 9.1). The goal is to evaluate how the 50,000 most correlated genetic variants, once taken together, are predictive for each ROI and to associate a statistical p-value measure with these prediction scores. We expressed this computation as 28,000 map tasks. Each map job performs 5 permutations, out of the 10,000 permutations required for each of the 14 ROI. Choosing to perform 5 permutations per map job was a pragmatic choice in order to keep the task timespan in the order of tens of minutes while maintaining a reasonable number of tasks. For example, executing 1 permutation per task, would reduce the timespan of a job to tens of minutes but would generate 140,000 jobs, which would create additional challenges to handling and monitoring their progress as discussed in Section 9.3.

This Big Data experiment was performed using the multi-site hierarchical MapReduce, introduced in Section 8. The framework was deployed on two Microsoft Azure data centers, the North-Central and West US. These sites were recommended by the Microsoft team for their large capacity. In each site, Azure storage service instances (i.e., Blob and Queue) were used for task communication and inter-site data exchanges. Considering the performance evaluation presented in Section 8.3 and the resource constraints of the algorithm, we chose to use the Large VM type, featuring 4 CPUs, 7 GB of memory and a local storage of 1 TB. Two deployments were set up in each cloud site, running 250 VM totalizing 1000 CPUs. Three of

ROI name		$CV\text{-}R^2$	fwe corr. p-value
Thalamus	left	**0.026**	1.10^{-4}
	right	**0.038**	1.10^{-4}
Caudate	left	**0.003**	2.10^{-4}
	right	-0.012	3.10^{-4}
Putamen	left	**0.019**	1.10^{-4}
	right	**0.006**	2.10^{-4}
Pallidum	left	**0.018**	1.10^{-4}
	right	-0.010	3.10^{-4}
Hippocampus	left	**0.010**	2.10^{-4}
	right	**0.020**	1.10^{-4}
Amygdala	left	**0.016**	1.10^{-4}
	right	**0.015**	1.10^{-4}
Accumbens	left	**0.022**	1.10^{-4}
	right	-0.002	2.10^{-4}



Results of the real data analysis procedure. (**Left**) predictive accuracy of the model measured by cross-validation, in the 14 regions of interest, and associated statistical significance obtained in the permutation test. (**Up right**) distribution of the $CV\text{-}R^2$ at chance level, obtained through a permutation procedure. The distribution of the max over all ROIs is used to obtain the family-wise error corrected significance of the test. (**Bottom right**) outline of the chosen ROIs.

Figure 9.1: The A-Brain analysis and its findings.

the deployments, each having 82 nodes, contained the TomusBlobs-MapReduce components of the hierarchical MapReduce. The fourth deployment had the role of partitioning the input data, action done by the Splitter, and aggregating the global result of the computation based on the Map-IterativeReduce technique. This latter phase consisted of 563 reduce jobs, each reducer processing 50 intermediate results (i.e., the Reduction Ratio parameter of the Map-IterativeReduce process is 50).

The experiment duration was 14 days, while the processing time for each of the 28,000 map jobs was approximately 2 hours. As generally acknowledged, failures are expected in large infrastructures such as the clouds and applications need to cope with this, especially for long running-time periods. During the experiment, the Azure services had an exceptional outage during which it was temporarily inaccessible [18], due to a failure of a secured certificate. In addition to this outage, only 3 regular VM failures (i.e., stop-restart) were encountered during this long run. The fault tolerance mechanisms provided by our solution managed to cope with these situations, enabling the analysis to be completed.The effective cost of this analysis was approximately equal to 210,000 compute hours, which cost almost 20,000$, accounting for the VM, storage and outbound traffic price.

Achievements. As a result of this joint collaboration and the effectiveness of our proposed solution to successfully execute this Big Data experiment, we showed how the functional signal in subcortical brain regions of interest can be significantly predicted with genome-wide genotypes. The importance of this discovery is amplified by the fact that this was the first statistical evidence of the heritability of functional signals in a failed-stop task in basal ganglia. This work had a significant impact in the area, being published in Journal of Frontiers in Neuroinformatics. Figure 9.1 presents the results of the corresponding correlation values for the significant links founded. This experiment demonstrates the potential of our approach for supporting Big Data applications executions at large-scale by harnessing the

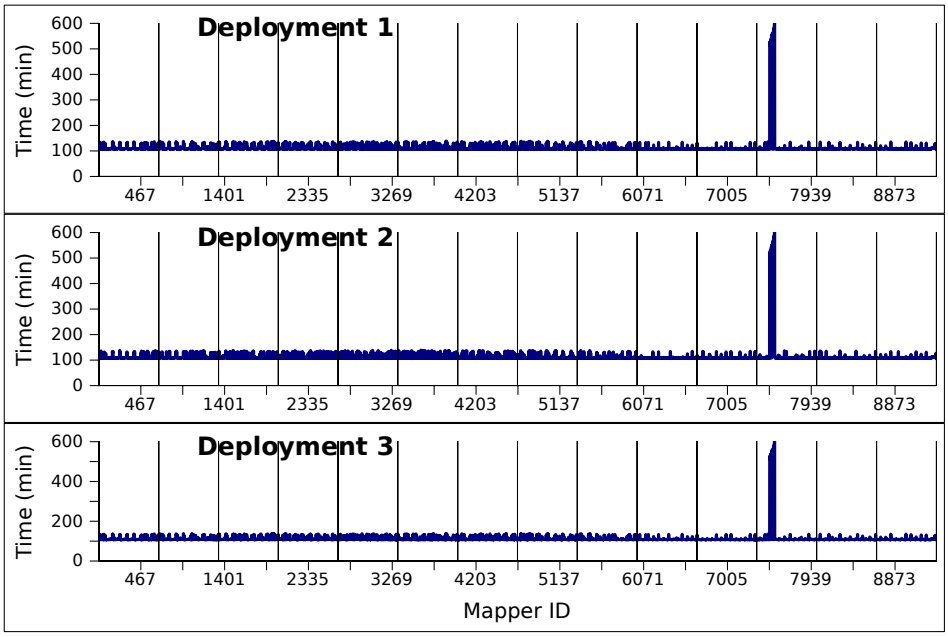


Figure 9.2: The execution of the compute tasks in 3 deployments. The pike variation corresponds to the cloud services becoming temporarily unavailable. 28,000 tasks are executed as map jobs equally partitioned between deployments

available computation power from multiple cloud data centers. Furthermore, it shows the benefits that our solution can bring for data science by enabling the migration of scientific applications to the cloud.

9.2 Focusing on Long-Running Scientific Experiments

We continue with an analysis of the performance aspects of this large-scale experiment. We start by presenting the execution time of the 28,000 map jobs in Figure 9.2. These are partitioned according to their execution in one of the 3 deployments running the TomusBlobs-MapReduce components of the hierarchical MapReduce framework. The first observation is that task timespans are similar for all the deployments, regardless the location of the deployments within the sites. This is achieved thanks to the initial partitioning done by the Splitter component (see Figure 8.5 in Section 8) and the local buffering of the data done by TomusBlobs, which spares mappers from remote data accesses.

Handling failures. The outage times that appear towards the end of the experiment are caused by the temporary failure of the Azure cloud, which made all cloud services inaccessible and thus isolated the compute nodes [18]. TomusBlobs reacted by suspending the mappers once the communication outside the VM was not possible. The mappers were kept idle during the outage period until the cloud became available again and the computation could be resumed. In addition to this exceptional failure, during the experiment, 3 map jobs were lost due to fail-stop VM failures. These jobs were restored by the monitoring and surveillance mechanism that we provided for the Splitter. The same mechanism was used

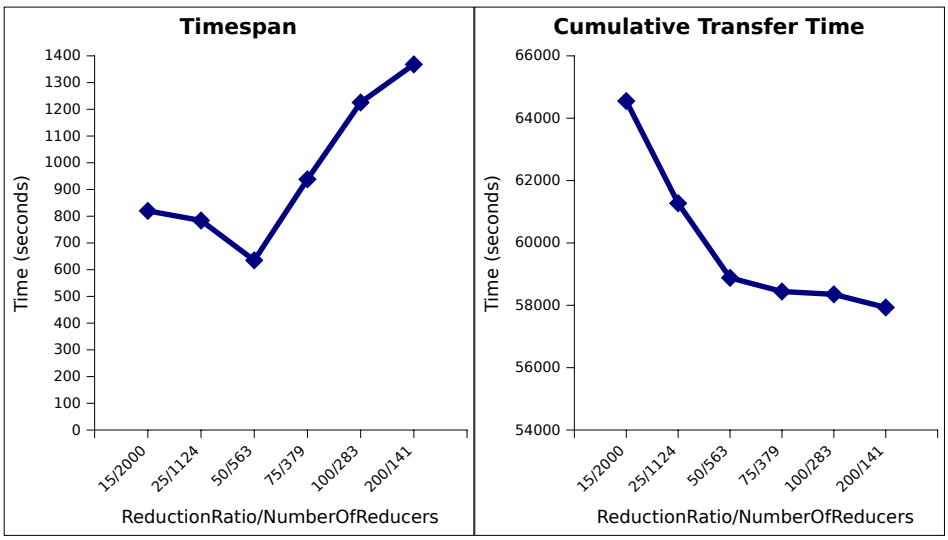


Figure 9.3: Overview of the IterativeReduce phase with respect to the number of reducers and the Reduction Ratio parameter of the Map-IterativeReduce technique. On the left chart, the timespan of the overall reduction process with 250 nodes; on the right chart, the cumulative transfer times.

also to renew the job descriptions, in order not to be discarded by the expiration mechanism of the Azure Queues. Currently all messages in the cloud queue are discarded after 1 week. By designing these mechanisms to ensure the completion of the analysis despite both regular failures and major outage, our solution demonstrated that it can provide high reliability even for geographically distributed computation.

Focus on the reduction phase. Next, we present in Figure 9.3 an analysis of the reduction phase considering the number of reduce jobs alongside with the corresponding reduction ratio (i.e, the number of intermediate results processed per reduce jobs). The goal is to study the relation between the compute phase of a reducer and its I/O phase (i.e., the acquisition of the data to be reduced). The parallel reduction process proposed by the Map-IterativeReduce technique brings significant improvements up to the point where the communication overhead becomes significant. Additionally, having a small reduction ratio translates to a large number of reducer jobs. This leads to jobs being executed in waves, which in turn increases the timespan of the reduction process. Making such an analysis before the actual experiment, allowed us to select the proper reduction ratio of 50, which helped decrease the overall execution time of the A-Brain analysis.

Focus on data management mechanisms. We continue the evaluation by studying the data space of the experiment, shown in Figure 9.4. The goal is to quantify the gains brought by our proposals in terms of transfer size reduction as well as to understand the strategies that one needs to setup for sparing the I/O volumes of an experiment. We start from a naive estimation of the overall amount of data which is obtained by multiplying the number of jobs with the data volume employed per job, i.e., input data size, output, environment setup and log file. Considering the large number of jobs of the A-Brain workload of 28,000, the

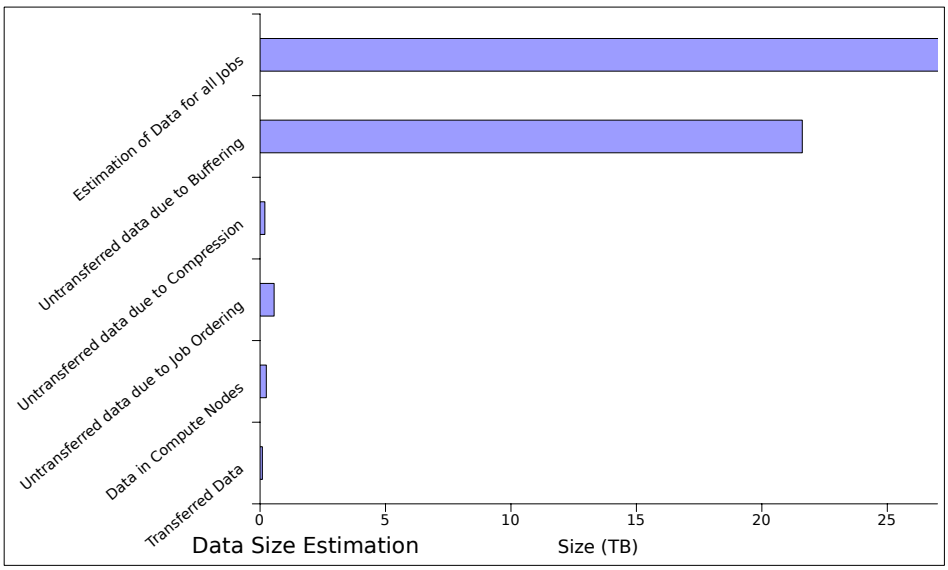


Figure 9.4: Estimation of the data sizes within the large A-Brain experiment. Each label depicts the corresponding data size for a particular scenario: total data space for the theoretical upper scalability bound; the amount of data spared from transferring due to buffering, compression or ordering of jobs; the data space of the problem with respect to the number of nodes used; the amount of transferred data.

data space reaches tens of terabytes, shown by the "Estimation of Data for all jobs" label in Figure 9.4. However, this scenario corresponds to all the jobs being executed in parallel on different nodes, which would require the maximum degree of redundancy for the data (e.g., the input data and the environment would have to be replicated on each node). Therefore, this estimation provides an upper bound in terms of data volume movements with respect to the maximum degree of parallelism. The cost of transferring such amounts of data would be enormous, both time- and money-wise. This is an important observation which shows that simply increasing as much as possible the parallelism degree of an application is not necessarily the best strategy in the cloud as it might be in HPC. This study suggests that a trade-off should be considered between the parallelism degree of the system and the data space. Apart from properly adjusting the number of processing nodes (i.e., the scale of the system), different techniques can be used to avoid furthermore data transfers between them. Buffering data, both at the level of the compute node, to reduce all communication, and at deployment level, to avoid costly inter-site communication, is a straightforward strategy to be used.The corresponding gain, marked in Figure 9.4 by the "Untransferred data due to Buffering" label, is substantial in terms of volumes of data spared to be exchanged.

Other techniques worth considering for such Big Data experiments range from data compression, enabled in our data-management system, to application-specific solutions like reordering the tasks. After applying all these techniques, the resulting data space for this experiment, with respect to the actual number of nodes used, is shown by the "Data in Compute Nodes" label in Figure 9.4. This analysis reveals two facts. On the one hand, a preliminary analysis of the application space is crucial to understand the options which can optimize the management of data. On the other hand, the data aspects (i.e., data space, redundancy generated by scaling, efficiency buffering) of the computation need to be consider

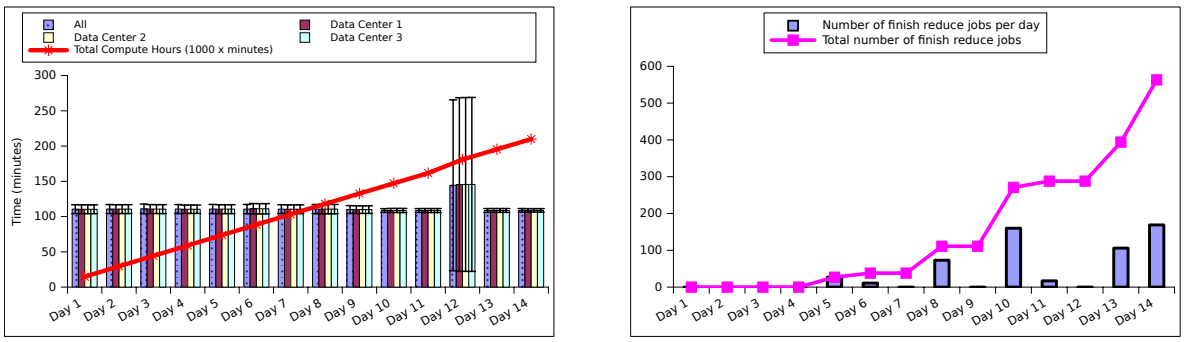


Figure 9.5: The A-Brain experiment progress. Left: The average map times per day and the corresponding standard deviation. Right: The compilation time of reduce jobs.

when scaling the parallel execution of scientific applications.

Stability and load balancing. The final step of our evaluation, shown in Figure 9.5, considers the timeline progress of the A-Brain analysis. In the left subfigure, we report the average map times within each day of the experiment and the corresponding standard deviation. Together with these, we report the aggregated *cloud compute hours* consumed by all the leased resources. These results show the stability and the predictability of the performance delivered by the TomusBlobs-based hierarchical MapReduce framework. In fact, the only noticeable shift in performance appears in the 12th day in which the cloud outage happened, as previously mentioned.

In the right subfigure of Figure 9.5, we show the progress towards computing the final unique result by the Reduce phase. It can be observed that as our approach eliminates the synchronization barrier between mappers and reducers, used by state-of-the-art MapReduce engines, the reduction process runs in parallel with the map computation. The irregular number of reduce jobs finished per day is due to the amount of intermediate results, given by the reduction ratio parameter of the Map-Iterative process, that a reducer must wait for, both from mappers or from other reducers. Considering that reducers tend to progress at the same pace (i.e., they have the same priority to acquire results), most of the jobs finish towards the end of the experiment, when most of the map jobs have completed. By enabling this behavior, we managed to balance the load across the compute nodes and to make the reducer computation non-intrusive to the overall execution.

9.3 Addressing Data Management Issues across Data Centers: Towards High-Performance Inter-Site Data Management

In the process of designing, implementing, supporting and validating our cloud storage solutions through large-scale experiments, we had the opportunity to gain useful experience and to learn several interesting lessons. This section presents these lessons. It also discusses issues that need to be addressed to continue to improve the support of running Big Data applications on the clouds. While some of these lessons specifically relate to the Azure cloud

platform, others can be considered from a more general perspective being relevant for any platform.

9.3.1 Azure-Specific Observations

Message visibility timeouts for Azure Queues. Azure Cloud Queue is offered by the vendor as the reliable communication mechanism to assign work to compute nodes in all processing scenarios. They differ from the traditional queue data structures as they lack the ability to ensure FIFO ordering. However, they guarantee that a message is delivered at least once. This implies potential duplication of messages, which is the result of the fault tolerance mechanism. Messages read from the queues are not deleted, but instead hidden until either they are explicitly deleted or until a certain timeout expires and they become visible again. TomusBlobs-MapReduce complements this timeout mechanism, used to guarantee the execution of the message containing job description, with a uniform naming schema to verify the potential duplications of results. However, messages are not held in the queue for unlimited time, but are discarded after a while, called visibility timeout. Initially set for 2 hours, the visibility timeout was increased to one week with the latest Azure API update. The reason why the visibility timeout matters is that if a workload is large it might take longer time to complete the processing (i.e., to execute all jobs of the workload). This is often the case with Big Data applications. However, in such long-running scenarios, queue messages containing job description are discarded by the cloud tool, which is supposed to guarantee the communication, leading to incorrect results. We faced this issue during the long-running A-Brain experiment. The solution we considered is to have an application-level tracking mechanism for the submitted message jobs. In our case this role was implemented by the Splitter component. The messages were monitored for their time to expiration and were resubmitted to the queue to reset their expiration deadline.

Application deployment-times. The cloud framework is in charge of the process of allocating the leased nodes and deploying the user application on them. While working with the Microsoft Azure cloud we have observed that this process plays a crucial role in the overall application performance. For each new or updated deployment on Azure, the fabric controller prepares the role instances requested by the application. This process is time-consuming and varies with the number of instances requested as well as with the deployment size (e.g., application executables, related libraries, data dependencies). The deployment times were reduced to a few minutes after the latest Azure update. However these times can still be a major bottleneck especially in the context of real-time processing scenarios (e.g., real-time resource provisioning). Other cloud platforms such as Amazon EC2 face similar or worse problems regarding the time for large resource allocation [102]. To minimize these deployment-times, one option is to build the environment setup at runtime from generic stubs and to improve the performance of multicast transfers between the physical nodes of the cloud infrastructure.

Handling VM failures. The general belief is that clouds are highly unreliable as they are built on top of clusters of commodity services and disk drives. Because of this, it is believed that they host an abundance of failure sources that include hardware, software, network connectivity and power issues. To achieve high-performance in such conditions, we had to

provision for failures at the application level in our initial estimations and designs. However, we discovered that only a very small fraction of the machines failed, even during the course of long-running executions; we recall that during the 2 week, 1000 core experiment, only 3 machines failed. Such an observation indicates that in fact clouds are more reliable that the general belief. This high-availability delivered by the cloud is due to the complex, multi-tiered distributed systems that transparently implement in Azure sophisticated VM management, load balancing and recovery techniques. We note from our experience that even though disk failures can result in data losses, we did not encountered any; making the transitory node failures to account for most unavailability. Considering these observations, it is a good strategy for Big Data processing frameworks to use and build on the fault-tolerant mechanisms already offered by the cloud software stack. The TomusBlobs-based approach does so by relying on Azure mechanisms to recover the compute nodes which is complemented by the watch-dog mechanism for logging/roll-backing the computation progress.

9.3.2 Beyond Azure

Wave-like versus pipelined MapReduce processing. In general, parallel tasks tend to be started at the same time on all available resources. Since usually the number of jobs is higher than the number of resources, such a strategy makes tasks to be executed in successive "waves". One lesson we learned while processing workloads with our MapReduce approach is that starting all the jobs (e.g., maps) at once is inefficient as it leads to massively-concurrent accesses to data. For example, concurrently retrieving the initial data by all nodes executing a task creates a high pressure on the cloud storage system. Additionally, the network bandwidth is inefficiently used, being either saturated or idle. The alternative we chose was to start the jobs in a pipeline manner, i.e., starting them sequentially as opposed to starting them all at the same time. Map tasks are created along the pipeline, as soon as their input data becomes available, in order to speed up the execution and to decrease the concurrency of accessing storage. Moreover, this approach allows successive jobs in the pipeline to overlap the execution of reduce tasks with that of map tasks. In this manner, by dynamically creating and scheduling tasks, the framework is able to complete the execution of the pipeline faster by better using the leased resources.

Data buffering and caching. Another lesson we learned from our analyses is that, during long-running experiments, many data blocks are likely to be reused. It then becomes useful to buffer them locally as much as possible in order to avoid further transfers. In fact, the analysis of the experiment showed that this strategy can bring the highest gains in terms of reducing the volumes of data transferred (in our case this was more than 70 %). Additionally, it is possible to leverage the data access pattern of applications to cache and transfer the data better, as shown with the adaptive mechanisms of TomusBlobs. The idea is to receive hints on the potential reuse of a data block, before taking the decision of discarding it. Such hints can be given by the components which deploy the workload into the system. In the case of our hierarchical MapReduce engine, the Splitter component is responsible for providing such hints.

Scheduling mechanisms for efficient data access. Due to a lack of studies regarding the best strategies for accessing data from geographical-distant cloud data centers, these oper-

ations tend to be done without any scheduling. However, in a cross data centers deployment, a high number of geographical distributed concurrent accesses are likely to lead to operation failures (e.g., TCP connections to the data source get closed). To exemplify and quantify this, we set 100 nodes to try to download the same data from a different geographical region, using the cloud storage API. Our initial empirical observations showed that up to 70 % of the read attempts fail, with the connection to the cloud storage being lost before the completion of the operation. A naive solution is to retry the operation in a loop until it succeeds. Besides being inefficient and generating additional failures, this strategy might actually lead to starvation for some of the nodes. Adding random sleeps between retries (as in the network decongestion algorithms) works in some cases, rather for short operations, but converges very slowly in general if there are many concurrent clients. We opted for a token-based scheduling approach, by allowing a predefined maximum number of nodes to access a shared resource concurrently. This solution provides several benefits. On the one hand, it allows the number of concurrent access to be calibrated with respect to any storage to eliminate failed operation. On the other hand, it can be combined with the pipeline approach for starting jobs in order to maximize the usage of the resources.

Monitoring services to track the real-time status of the application. Scientists or system administrators of long-running applications require tools to monitor the progress of computation and to assert the performances of the underlying infrastructure. These operations are implemented using logging mechanisms (e.g., log files) in which the state of the application and the resources is registered. Currently, these log data are handled using the default cloud management tools (i.e., applications write the log data into cloud storage BLOBs). However, a practical lesson that we learned is the importance of a specialized monitoring system. Moreover, a critical feature is to extend this system to estimate the I/O and storage performance accurately and robustly in a dynamic environment. Such estimations are essential for predicting the behavior of the underlying network and compute resources in order to optimize the decisions related to storage and transfer over federated data centers. The challenge comes from the fact that estimates must be updated to reflect changing workloads, varying network-device conditions and multi-user configurations. To address this, a *Monitor as a Service* for the cloud needs to be provided. Building on this, data management systems can be designed for context-awareness which in turn maximizes performance, makes the system autonomous and minimizes cost. This is a way to take users needs and policies into account.

Going beyond MapReduce: cloud-based scientific workflows. Although MapReduce is the "de-facto" standard for cloud data processing, most scientific applications do not fit this model and require a more general data orchestration, independent of any computation paradigm. Executing a scientific workflow in the cloud involves moving its tasks and files to the compute nodes, an operation which has a high impact on performance and cost. As pointed out in the previous sections, the state-of-art solutions currently perform these operations using the cloud storage service. In addition to having low throughput and high latencies, these solutions are unable to exploit the workflow semantics. Therefore, it becomes a necessity to schedule the task and the data, within the compute VMs, according to the data processing flow, by considering the data layout, the allocation of tasks to resources and the data access patterns. So far, we have shown that it is possible to increase the performance of managing workflow data, within a data center, by considering the access patterns (see Sec-

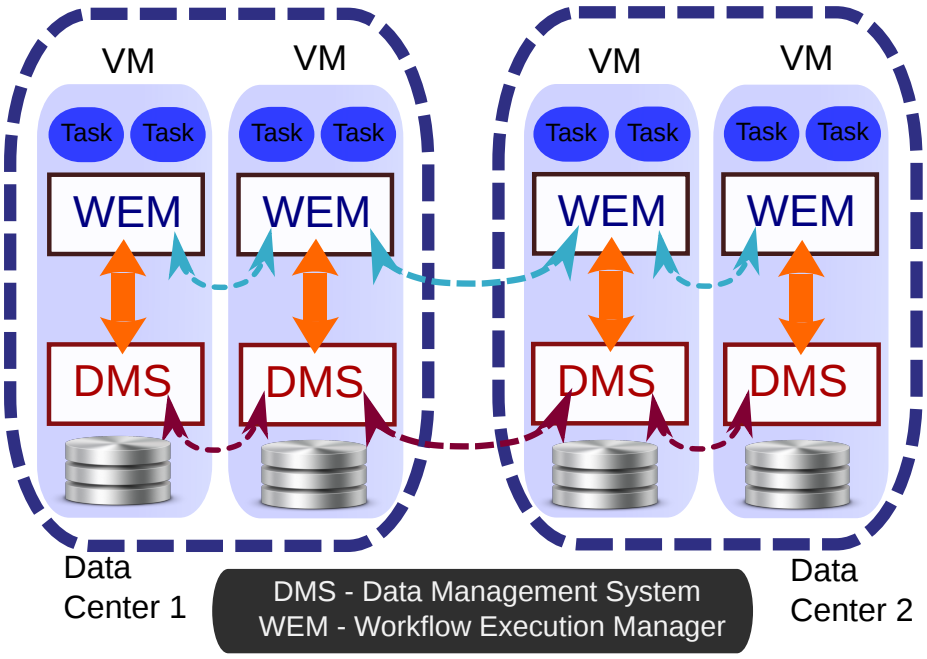


Figure 9.6: Cloud deployments containing a workflow engine and a data management co-scheduling the data and task executions

tion 7). This approach can be leveraged on the one hand for workflows executed across sites, and on the other hand to further interconnect the scheduling of tasks (handled by the workflow engine) with the data (handled by the data management system). In order to achieve this, we propose a two-way communication between the workflow engine (also referred to as workflow execution manager - WEM) and the data management system (DMS), as illustrated in Figure 12.8. This design, which interconnects the two components, enables them to collaborate in order to optimize the computation. To this purpose, scheduling data and tasks can be done by considering information about the other. Additionally, the critical decision of migrating either data or task from one data center to another can be taken dynamically, at runtime, according to the cloud environment.

Towards high-performance Geographically Distributed Data Management. *The most important lesson we learned, is the lack of support of current cloud data management systems for handling efficiently data across cloud sites*. The critical issues we identified are the high latency, low interconnecting bandwidth between sites, the large variability and high costs. For now, direct transfers are not supported. The workaround, currently available only on top of the cloud storage service, is inefficient and involves persistently storing data (the cloud storage is optimized for persistent and long term storage). Thus, scientists, most commonly non-cloud experts, face the challenge of designing their own data management tools for supporting even basic data exchange operations across their application instances. This leads to inefficient, non-scalable and sometimes faulty solutions (e.g., solution can easily neglect important aspects regarding fault tolerance or consistency guarantees under concurrency). Even for cloud experts, designing high-performance architectures for managing data across sites is not trivial, since there is little knowledge about the effectiveness of data management

options or strategies for such transfers. Overcoming these challenges is a key milestone for addressing the future challenges brought by the Big Data era. We bring a contribution in this direction by designing a *Transfer as a Service*, coupled with a dynamic cost model, which enables applications to transfer data efficiently, while minimizing the costs. This solution together with the underlying approaches are discussed in Part III of the thesis.

9.4 Discussion

Porting Big Data applications to the clouds brings forward many issues regarding how cloud infrastructures can be used most efficiently for solving these problems. In this landscape, building high-performance systems to address the requirements of these applications is critical and challenging. Efficient storage and scalable parallel programming paradigms are some critical examples. TomusBlobs addresses these issues and as a result, the scientific discovery can be accelerated using the cloud. We demonstrated the benefits of our approach through a multi-sites experiment on a thousand cores across 3 Azure data centers, while consuming more than 200,000 compute hours - one of the largest scientific experimental setup on Azure up to date. The experiment showed that our approach is fault tolerant and it is able to provide reliable performance at large-scale while running the computation across multiple sites. On the application side, the analysis enabled by our solution discovered how the functional signal in subcortical brain regions can be significantly fit with genome-wide genotypes.

Evaluating the cloud landscape using real Big Data application scenarios taught us important lessons. Cloud systems need to monitor and account for the cloud variability. This context information needs to be leveraged in the decision mechanisms regarding the management and processing of data. Scheduling strategies are required both for task and for data transfers. Finally, as the processing systems scale according to Big Data workloads, the existing cloud model in which all types of data management exclusively rely on the cloud storage becomes widely inefficient and obsolete. Therefore, it is critical to enable specialized cloud solutions for high-performance geographically data management. Moreover, with the growing proportions of the real-time data in the Big Data ecosystem, stream data management for clouds needs to be considered as well. Currently there is no support for cloud streaming let alone high-performance streaming across data centers. We address these issues related to high-performance data management, both static and real-time, across cloud data centers in Part III of this thesis.

Part III

High-Performance Big Data Management across Data Centers

Chapter 10

DataSteward: Using Dedicated Nodes for Scalable Data Management

Contents

10.1 A Storage Service on Dedicated Compute Nodes 107
10.2 Zoom on the Dedicated Node Selection 110
10.3 Experimental Evaluation and Functionality-Perspectives 113
10.4 Discussion 120

This chapter develops the contributions published in the following paper and book chapter:

- *DataSteward: Using Dedicated Nodes for Scalable Data Management on Public Clouds* Radu Tudoran, Alexandru Costan, Gabriel Antoniu. In Proceedings of the 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (ISPA/TRUSTCOM 2013), Melbourne, Australia, July 2013, pp.1057-1064
- *Big Data Storage and Processing on Azure Clouds: Experiments at Scale and Lessons Learned*. Radu Tudoran, Alexandru Costan and Gabriel Antoniu.In the book *Cloud Computing for Data-Intensive Applications*, to be published by Springer, 2015. Editors Xiaolin Li and Judy Qiu

One of the critical needs for dealing with the "data deluge" is to obtain an efficient, scalable and reliable storage for cloud applications. In Part II of this thesis, we discussed and illustrated the limitations of the cloud storage service in the context of Big Data processing. Our observations emphasized that using the cloud storage is highly inefficient. We

also showed that it cannot cope with all application needs, as it trades performance for data durability. This represents a major limitation, as highlighted in Section 9, which prevents global scaling of applications (i.e., processing data across cloud data centers). Additionally, applications using the cloud storage remain unresponsive, in their data handling mechanisms, to any changes in the VMs performance. Monitoring resources and handling data in an environmentally-aware manner are key milestones towards high-performance and cost efficient Big Data analysis.

Collocated Storage. The alternative to the cloud storage that we have introduced, Tomus-Blobs, federates the local virtual disks of the compute nodes into a globally shared storage while leveraging data-locality. Applications use a typical cloud API (i.e., data handling is done via the same functions as accessing Azure Blobs or Amazon S3 cloud storage) to store, checkpoint and access large intermediate data. However, executing Big Data applications requires more advanced functionalities to be provided by the data management system such as logging mechanisms, compression or transfer capabilities. Building such features on top of the local collocated storage can become rather intrusive and impact on the application's perceived performance. Besides, this collocation strategy provides a higher throughput than the cloud storage, but it is subject to application-level failures which can stop the host nodes. An unhandled runtime-exception in the application, a memory overflow, a corrupt type as well as many other errors will lead to a VM reboot. Such an event would translate into a degradation of the storage performance. Moreover, there is a direct correlation between the CPU usage and the data throughput [181], which can introduce performance bottlenecks that slow down either the computation or storage access.

Our approach in a nutshell: Providing Advanced Data Stewarding on Dedicated Nodes. To deal with these limitations we propose DataSteward, a data management system that provides a high degree of reliability while remaining non-intrusive through the use of dedicated compute nodes. This separation allows applications to perform I/O operations efficiently as data is preserved locally, in storage nodes selected within the application deployment. The idea of using dedicated infrastructures for performance optimization is currently explored in HPC environments. Dedicated nodes or I/O cores on each multicore node are leveraged to efficiently perform asynchronous data processing, in order to avoid resource contention and minimize I/O variability [50]. However, porting this idea to public clouds in an efficient fashion is challenging, if we consider the multi-tenancy model of the cloud, the resulting performance variability of resources and the use of unreliable commodity components. To address this challenge, DataSteward selects a set of dedicated nodes to manage data by using a topology-aware clustering algorithm. To capitalize on this separation, we introduce a set of scientific data processing services on top of the storage layer, that can overlap with the executing applications. The insight of this approach is similar to the concept of a file, from a traditional computer, which is a generic object associated with a set of operations (e.g., move, view, edit, delete, compress), therefore a "cloud file" requires its own set of actions. To validate this approach, we perform extensive experimentation on hundreds of cores in the Azure cloud, which show an improvement in the overall performance up to 45 %. Finally, by decoupling the storage from the applications, DataSteward increases security by isolating data from potential malicious attacks on the web service.

One of the key aspects that determines the application performance is the *selection method*

of the dedicated nodes. Others have also recognized the need to make optimal cloud application deployments as a challenging research problem. Based on the cloud nodes' QoS performance, a number of selection and scheduling strategies have been proposed in the recent literature. The major approaches can be divided into three categories. *Random* methods select nodes randomly and have been extensively employed in works like [11]. *Ranking* strategies rate the cloud nodes based on their QoS and select the best ones [5, 33]. A relevant example is NodeRank [42], which, inspired by Google's PageRank, applies Markov random walks to rate a network node based on its resource and the importance of its links with other node. The importance refers here to the relative resource quality of a node. In general, ranking methods only use QoS values and omit to consider the communication performance between nodes, making them improper for data intensive workloads or for storage-related scenarios. *Clustering* based methods like [55, 56], consider the communication between the selected nodes. The basic idea of these strategies is to cluster the cloud nodes that have a good communication performance together, to deploy an application.

Our approach in a nutshell: Selecting the Storage Nodes. We use a similar approach to cluster all the nodes within a deployment and then choose "leaders" from each cluster (those best connected with all nodes within the cluster), which will make up the global dedicated storage nodes. In contrast to existing clustering solutions, we first discover the communication topology and the potential virtual network bottlenecks by pre-executing a series of measurements. Next, we consider both the resource and the topology properties of a node in a unified way, in order to select the most suited ones. This is a key difference from existing works, which only take into consideration the CPU and/or the bandwidth of a node. Compared to this state-of-the-art node selection algorithms, our approach brings up to a 20 % higher throughput. Moreover, the solution is generic, being applicable in other cloud scenarios, allowing both users and cloud providers to obtain higher performance from cloud services based on nodes selection.

10.1 A Storage Service on Dedicated Compute Nodes

The selection of the storage option, that supports the computation, is a key choice as it impacts both the performance and the costs. The standard cloud offering for sharing application data consists of storage services accessed by compute nodes via simple, data-centric interfaces which only providing *read/write* functionality. The alternative option to the cloud object storage is a distributed file-system deployed on the compute nodes, such as the TomusBlobs approach, that we introduced in Section 7. Each one of these options are geared for various types of data and maximize a different (typically conflicting) set of constraints. For instance, storing data locally within the deployment increases the throughput but does not isolate the storage from computation, making it subject to failures or performance degradation. Using the cloud storage provides persistence at the price of high latencies and additional costs. Moreover, these services, focus on data storage primarily and support other functionalities, such as inter-site transfer or logging, essentially as "side effects" of their storage capability. However, enabling the execution of Big Data applications on the cloud requires advanced data centric services, as emphasized by our analysis in Section 9. To properly address these issues we introduce in this section DataSteward. This data management

scheme extends the list of design principles introduced for TomusBlobs in Section 7 with 3 additional ones.

Dedicated compute nodes for storing data. This approach preserves the data proximity within the deployment and increases the application reliability through isolation. The goal is to have high data access performance, as provided by TomusBlobs through data locality, while freeing application nodes from managing data, as when using the remote cloud service. Moreover, keeping the management of data within the same compute infrastructures (i.e., same racks, switches) optimizes the utilization of the cluster bandwidth by minimizing the intermediate routing layers to the application nodes. Finally, this principle makes our solution applicable to any cloud scenario, including resource-sensitive applications for which sharing part of the compute resources (e.g., memory or CPU) would not be possible.

Topology awareness. Users applications operate in a virtualized space which hides the infrastructure details. However, the proposed data management solution needs some mechanism to ensure that the dedicated storage servers are located as "close" as possible to the rest of the computing nodes. This "closeness", defined in terms of bandwidth and latency, determines the communication efficiency. As information about the topology is not available, our solution estimates it, based on set of performance metrics, in order to perform an environment-aware selection of nodes.

Overlapping computation with data processing. With this dedicated node approach, we are able to propose a set of advanced data processing services (e.g., compression, encryption, geographically distributed data management, etc.). These are hosted on the dedicated nodes and address the needs of scientific Big Data applications. In a typical cloud deployment, users are responsible for implementing such higher level functionality for handling data, which translates in stalling the computation for executing these operations. Our approach offloads this overhead to the dedicated nodes and provides a data processing toolkit as well as an easily extensible API.

DataSteward is designed as a multi-layered architecture, shown in Figure 10.1. It is composed of 3 conceptual components:

The Cloud Tracker has the role to select the nodes to be dedicated for the data management. The selection process is done once, in 4 steps, at the starting of the deployment. First, a leader election algorithm is run, based on the VM IDs. Second, the trackers within each VM collaboratively evaluate the network links between all VMs and reports the results back to the leader. Third, the leader runs the clustering algorithm described in Section 10.2 to select the most fitted, throughput-wise, nodes for storage. Finally, the selection of the nodes is broadcast to all compute nodes within the deployment. The Cloud Trackers evaluate the network capacities of the links, by measuring their throughput, using the `iperf` tool [99]. The communication between the trackers and the elected leader is done using a queueing system. The same communication scheme is used by the leader to advertise the dedicated nodes to all VMs.

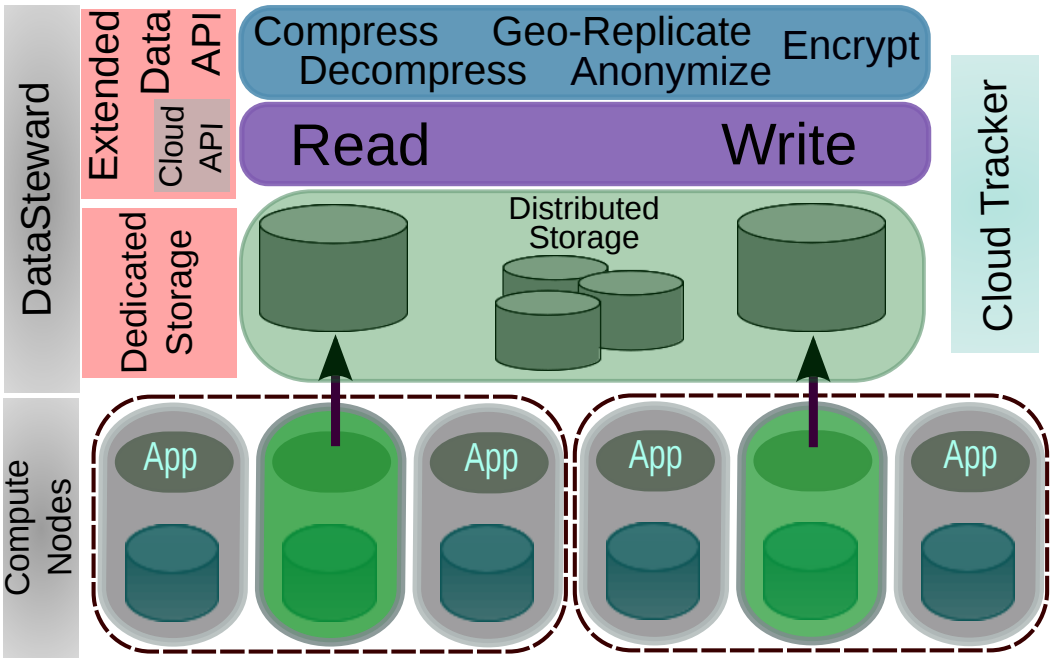


Figure 10.1: DataSteward architecture. Compute nodes are dedicated for data storage, on top of which a set of data processing services are provided.

The Distributed Storage is the data management system deployed on the dedicated nodes, that federates their local disks. Users can select the distributed storage system of their choice, for instance any distributed file system that can be deployed in a virtualized environment. This property together with providing data locality, represent common principles shared with the Storage Entity of TomusBlobs, described in Section 7.1. However, the Distributed Storage is now decoupled from the main computation, protecting the data from potential application-level failures. Additionally, the local memory of the dedicated nodes is aggregated into an *in-memory storage*, used for storing, caching and buffering data. The Distributed Storage can dynamically scale up and down, dedicating new nodes when faced with data usage bursts or releasing some of the existing ones. The implementation of the prototype is done in C#, while the validation is done using the Azure cloud. Finally, for evaluating the system and for a fair comparisons with the TomusBlobs approach, discussed in Section 10.3, we use the BlobSeer backend for both approaches.

The Data Processing Services are a set of advanced data handling operations, provided by DataSteward and targeting scientific applications. The goal is to capitalize the computation power now available for data management, in order to provide high-level data functions. Such functionality, exposed to applications and provided directly at the level of the storage system, increases the performance of handling data and extends the basic cloud data API which exists today. Examples of such advanced functions include: compression, geographical replication, anonymization, as well as other which are further discussed in Section 10.3.4. The Data Processing Services are exposed to applications through an API, currently available in C#, independent of the distributed storage chosen by the user. The API extends the

default cloud-based data interface, which only provides read and write operations. At the same time, it keeps the handling of data transparent to applications. In this way, the system remains generic and is able to accommodate future Big Data requirements. To this end, the data processing services are loaded dynamically, from the default modules or from libraries provided by the users. We present bellow three implementations of such services, while others are described in the future chapters.

Geographical replication. Whether to achieve high fault tolerance, for interoperability with other web-services or for disseminating and sharing the applications results, this is a useful service in the context of federated clouds. As the involved data movements are time and resource consuming, it is inefficient for applications to stall their executions in order to perform such transfers. DataSteward provides an alternative, as the applications can simply check-out their results to the dedicated nodes, which acts as a broker. This operation is fast, consisting in a low latency data transfer within the deployment. Then, DataSteward performs the time consuming geographical replication, while the application continues the main computation.

Data compression. Typically, the separation of scientific applications in multiple tasks leads to multiple results. Before storing them persistently, one can decrease through compression the incurred costs of persistent (long-term) storage. By grouping together these results on the dedicated nodes, we are able to achieve higher compression rates, than if the results were compressed independently on their source node. In fact, many scientific applications have been shown to have high spatial or time correlation between the outputs of the computing sub-processes [69, 130]. To this end, DataSteward exploits these data similarities and minimizes the compression overhead of multiple files/objects, reaching compression rates that could not be achieved otherwise at the process or node level.

Scientific data processing toolkit. Scientific applications typically require additional processing of their input/output data, in order to make the results exploitable. For large data sets, these manipulations are time and resource consuming. Moreover, an aspect which typically tends to be neglected, but impacts the overall data handling time as well, is the number of objects in the data set (e.g., I/O files, log files). Due to the simplicity of the default cloud storage API, these objects need to be managed independently as there are no operations which can be applied on multiple files. Additionally, they need to be downloaded at the client side most of the times, even for simple operations such as searching or checking the file exists. By using the dedicated nodes, such processing on multiple files can be provided directly at the data management layer and can be overlapped with the main computation. Therefore, DataSteward provides an integrated set of tools applicable on groups of files. These operations support file set transformations such as filter, grep, select, search or property check.

10.2 Zoom on the Dedicated Node Selection

As the selection of the storage nodes can significantly impact application performance, we believe that the topology and utilization of the cloud need to be carefully considered to come up with an optimized allocation policy. Since cloud users do not have fine-grained visibility

Algorithm 2 Initialization of the clustering algorithm.

Input:
Nodes $= \{\text{node}_1..\text{node}_N\}$ ▷ the set of compute nodes
$\text{Client}_{\text{Clusters}}[] = \{List_1..List_{\text{NrOfDataServers}}\}$ ▷ the set clients grouped in clusters
Output:
$\text{Servers} = \{\text{server}_1..\text{server}_{\text{NrOfDataServers}}\}$ ▷ the set of data servers - the cluster centroids
clients_per_server $= N/\text{NrOfDataServers}$

for $i \leftarrow 0, \text{NrOfDataServers}$ **do**
    Servers $\leftarrow$ node $\in$ Nodes (random selected)
end for

into or control over the underlying infrastructure or resources placement, they can only rely on some application-level optimization. In our case, the storage nodes are selected based on the topology that we, as cloud users, discover. The criteria that we want to maximize is the aggregated throughput that the application nodes will get to the dedicated storage nodes. To obtain an intuition of the cloud topology and the physical placement of the VMs, we propose to use a *clustering algorithm*. Our proposal relies on the fact that the cloud providers distribute the compute nodes in different fault domains (i.e., behind multiple rack switches). Hence, we aim to discover these clusters based on the proximity that exists between the nodes within a fault domain. To this purpose, we fitted the clustering algorithm with adequate hypotheses for centroid (i.e., the object used to represent a cluster) selection and assignment of nodes to clusters, in order to maximize the data throughput for our usage scenario. Finally, the selection of the dedicated nodes is done based on the discovered clusters. With this approach, we increase reliability by distributing the storage nodes across all the fault domains where the application runs. Furthermore, the compute nodes can perform all data operations within the fault domain, minimizing the overall data exchanges across switches and long-wires and thus avoiding throttling the network.

Clustering algorithms are widely used in various domains ranging from data mining to machine learning or bio-informatics. Their strength lies in the adequate hypotheses used for creating the clusters (i.e., the basis on which an element is assigned to a cluster) and for representing them (i.e., the hypothesis for selecting the centroids that represent the clusters). In our scenario, the clusters are formed from the compute nodes within the deployment, depicting the topology of the nodes across fault domains. Each such cluster is represented by the nodes selected to be dedicated for storage (i.e., the centroid). Our idea for the assignment of a node to a cluster is to use the throughput of the compute node to the storage node that represents the cluster. The working hypothesis is that nodes within the same fault domain will have higher throughput one to another because the interconnecting link does not cross any rack switches. Therefore, with this approach we are grouping the nodes and balancing the load generated by the compute nodes. The criteria to assign a node to a cluster is given in Equation 10.1.

$$cluster = \underset{i \in Servers}{\arg\max} \operatorname{Max} throughput[\underbrace{i, j}_{|Client[i]|<clients\_per\_server}] \tag{10.1}$$

Algorithm 3 Clustering-based dedicated nodes selection

```
1: procedure DEDICATENODES(NrOfDataServers, N)
2:     repeat
3:         changed ← false
4:                 ▷ Phase 1: Assign nodes to cluster based on proximity within clients limit
5:         for i ← 0, N do
6:             if i ∉ Servers then
7:                 max ← 0
8:                 maxserver ← 0
9:                 for j ← 0, NrOfDataServers do
10:                    if throughput[Servers[j], i] > max && Client[j].Count < clients_per_server then
11:                        max = throughput[Servers[j],i]
12:                        maxserver = j
13:                    end if
14:                end for
15:                Client[maxserver].Add(i)
16:            end if
17:        end for
18:
19:        ▷ Phase 2: Centroid Selection — reselect the data servers based the assignment of nodes to clusters
20:        for i ← 0, NrOfDataServers do
21:            maxserver ← 0
22:            max ← 0
23:            for all j ∈ Client[i] do
24:                if j.std < ADMITTED_STD and j.thr > ADMITTED_THR then
25:                    server_thr ← 0
26:                    for all k ∈ Client[i] do
27:                        server_thr + = throughput[j,k]
28:                    end for
29:                    if server_thr > max then
30:                        max ← server_thr
31:                        maxserver ← j
32:                    end if
33:                end if
34:            end for
35:            if Servers[i] ≠ maxserver then
36:                Servers[i] ← maxserver
37:                changed ← true
38:            end if
39:        end for
40:    until changed == true
41: end procedure
```

The next step of a clustering algorithm, after creating the clusters, is to update the centroids. The hypothesis that we propose is to select as a centroid the node towards which all other nodes in the cluster have the highest aggregated throughput. With this approach, we maximize the overall throughput achieved by the application VMs within the cluster to the storage node. Equation 10.2 shows the principle based on which the clustering algorithm updates the centroids.

$$maxserver = \arg\max_{j \in Client[i]} \sum_{k \in Client[i]} throughput[j,k] \tag{10.2}$$

Next, we present the cluster-based algorithm for selecting the dedicated nodes, which integrates the proposed hypotheses. In Algorithm 2 we introduce the data structures used to represent the problem (i.e., the set of compute nodes, the set of dedicated nodes) and the random initialization of the centroids. The advantage of starting with a random setup is that no extra information is required. Algorithm 3 describes the 2 phases of the clustering algorithm. The first phase corresponds to Equation 10.1. The compute nodes are assigned to the clusters based on the throughput towards the dedicated node and by considering an upper-bound for concurrency (i.e., the maximum number of clients allowed per server). Having an upper limit for the load per storage node, permits to apply this algorithm even for cloud platforms which do not guarantee a distribution of nodes across racks. Such scenarios, which would lead otherwise to unbalanced clusters, would be easily solved by our solution by partitioning the nodes based on their proximity within the rack. The second step, consists in updating the centroids. We select the nodes which provide the highest aggregated throughput within each cluster, according to Equation 10.2. At the same time, we filter the nodes with poor QoS (i.e., low throughput or high I/O variability). We introduce this filtering such that the system delivers a sustainable performance in time, as required by scientific computation.

10.3 Experimental Evaluation and Functionality-Perspectives

This section presents the performance evaluation of our approach. We perform the analysis both in synthetic settings and in the context of scientific applications. The experiments are carried out on the Azure cloud in the North Europe and West US data centers. The experimental setup consists of 50 up to 100 Medium Size VMs, each having 2 virtual CPU which are mapped to physical CPUs, 3.5 GB of memory and a local storage of 340 GB. The evaluation focuses on 3 aspects. First, we demonstrate the efficiency of the node selection strategy. Secondly, we compare the throughput performance delivered by DataSteward with the ones of Azure Blob storage and the TomusBlob approach. For the fairness of the comparison, the BlobSeer backend is used both for DataSteward and for TomusBlobs. Finally, we analyze the gains, in terms of application time, that the data services enabled by DataSteward brings for scientific computation.

10.3.1 Clustering Algorithm Evaluation

One of our key objectives was to design an efficient scheme for *selecting the dedicated storage nodes*. In the next series of experiments, we compare our selection approach with the state of

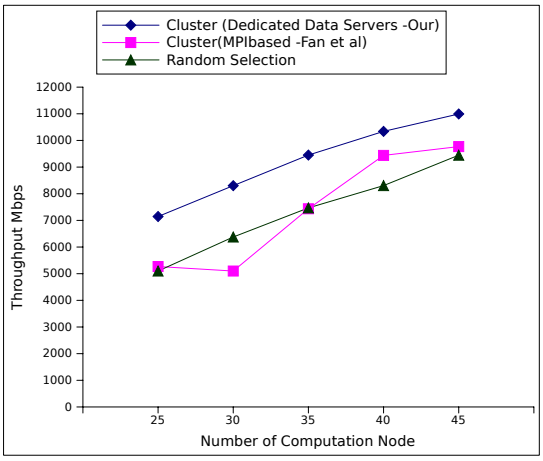


Figure 10.2: The cumulative throughput of accessing the storage deployed on dedicated nodes. The experiment varies the number of data servers (i.e., the dedicated nodes) and compute nodes (i.e., the application nodes accessing data).

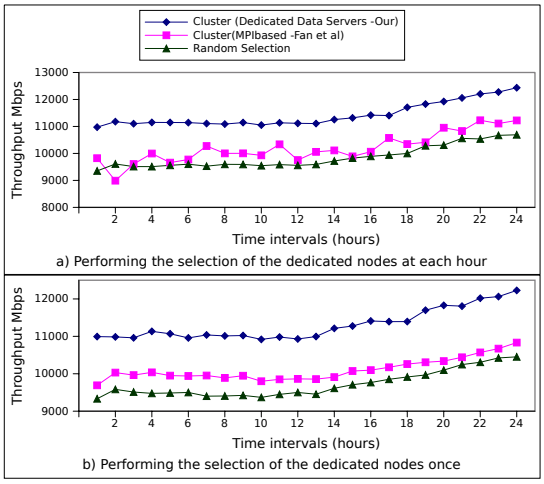


Figure 10.3: The cumulative throughput of the client within compute nodes when accessing the storage nodes. The storage nodes are reselected every hour (a - top) or selected only once (b - bottom).

the art clustering algorithm introduced by Fan et al [56], which clusters nodes together based on their QoS, and with a random selection that does not consider the underlying topology. The goal is to validate that our node selection strategy can indeed capture and leverage the infer communication topology within a deployment. The analysis is done using application throughput as a metric. The experiment methodology consists in dividing the nodes within the deployment in two groups: the compute nodes, where the application is executed, and storage nodes, where DataSteward is deployed. All measurements evaluate the performance achieved by the application compute nodes to access the storage nodes.

Impact of the compute/storage nodes ratio on applications I/O performance. The goal of the first experiment is to analyze how the data access performance varies for different ratios between compute and storage nodes. To this purpose, we fix the total number of nodes to 50. Then, the number of dedicated nodes is decreased progressively which in turn increases the number of compute nodes. The results of this experiment are shown in Figure 10.2, which depicts the cumulative throughput within the system. The first observation is that our approach is better suited for this scenario, achieving higher performances. A second observation is that the average throughput *per node* decreases when less nodes are dedicated for the storage. Nevertheless, the decrease throughput rate is inferior to the rate at which the clients are increased, which translates to a higher cumulative throughput for more compute nodes. This shows that our approach for dedicating storage nodes is able to cope with increasing number of clients, showing up to 20 % higher *cumulative throughput*, compared to the other selection strategies.

Focusing on when to select the storage nodes. Next, we assess how our decision of running the selection algorithm once, at the start of the deployment, impacts on the overall system performance. Avoiding to reconfigure the system is important as any change of a

data node would incur expensive data movements due to the large data volumes which can be stored by a node at a given time. For this analysis, we also rely on the aggregated throughput metric as before, but consider a 24-hour interval. We compare two time-wise options of applying the clustering and selecting the nodes: 1) *the node selection is repeated every hour* based on new network measurements and 2) the selection is *done only once* (i.e., our choice), at the beginning of the experiment. The results of this comparison are shown in Figure 10.3, with the hourly reselection approach on the top and the single selection at the bottom. One can observe that in both scenarios our clustering algorithm for node selection delivers slightly similar performance. This comes as a result of disseminating the storage nodes across the switches and racks, which makes our approach less sensitive to network variations. This is complemented by filtering the problematic VMs, with a high variability as shown in Algorithm 3, which further stabilizes the performance. This is a very important observation as it validates our choice of performing the node selection only once. Finally, in both scenarios our clustering algorithm outperforms the other strategies with more than 10%, capitalizing on its topology-awareness. These results show that our approach of selecting the nodes delivers superior and more stable performance than state of the art solutions.

10.3.2 Data Storage Evaluation

Our next series of experiments analyze the performance of the DataSteward storage layer. First, we compare its performance with TomusBlobs. The goal is to evaluate the benefits of dedicating compute nodes for storage against collocating storage with the computation. To ensure a fair comparison each system is deployed on 50 compute nodes and both use the BlobSeer backend (see Section 7.1). Secondly, the performance of DataSteward is also compared against the one of the cloud storage, e.g., the local cloud storage service (Local AzureBlobs) and a geographically remote storage instance from another data center (Geo AzureBlobs). The goal of this second comparison is to demonstrate that, despite DataSteward is not collocating data with computation, the delivered I/O throughput is superior to a typical cloud remote storage.

Scenario 1: Multiple reads / writes. We consider 50 concurrent clients that read and write, from memory, increasing amounts of data, ranging between 16 to 512 MB. When using TomusBlobs, the clients are collocated on the compute nodes, with 1 client in each node. For DataSteward the clients run on distinct machines, with one client in each machine as for TomusBlobs. We report the cumulative throughput of the storage system for the read and write operations in Figures 10.4 and 10.5, respectively.

Both approaches considerably outperform, between 3x and 4x, the cloud storage, whether located on-site or on a geographically distant site. Clearly, keeping data within the deployment has a major impact on the achieved throughput. If for TomusBlobs such results were expected, being consistent with the ones in Section 7.3.1, for DataSteward they confirm that our approach for dedicating nodes is superior, performance-wise, to a remote cloud service. This comes as a result of minimizing the number of infrastructure hops (e.g., switches, racks, load balancers) between client application nodes and storage node due to our topology-aware strategy for allocating resources. Hence, this strategy could be used also by the cloud provider to distribute its cloud storage nodes among the compute nodes within data centers.

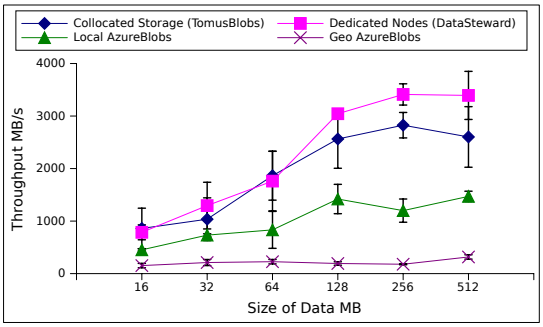


Figure 10.4: The cumulative read throughput with 50 concurrent clients.

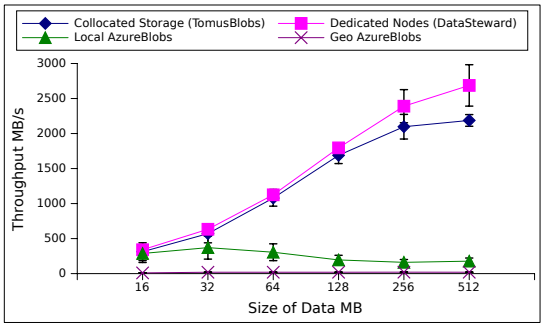


Figure 10.5: The cumulative write throughput with 50 concurrent clients.

Next, we zoom on the performance of DataSteward and TomusBlobs. One might expect that, due to data locality, the collocated storage option delivers better performance than managing data in dedicated nodes. However, the measurements reported in Figures 10.4 and 10.5 show differently. On the one hand, for small volumes of data handled by clients, the performance of the two approaches are similar. On the other hand, when the data sizes are increased, DataSteward outperforms the collocated storage with more than 15 %. This is due to 2 reasons. First, for both approaches, the underlying BlobSeer backend splits data into chunks scattered across the federated virtual disks; so even with the collocated storage not all data accessed by a client is always entirely present on the local VM. Moreover, the throughput is determined both by the network bandwidth, which is the same in both setups, and by the CPU's capability to handle the incoming data. The later is better leveraged by DataSteward which separates computation from communication. Secondly, with increasing sizes of data operated by a client, a poor management of the network links between the VMs leads to a faster saturation of the network. This can be observed for sizes beyond 200 MB, for which TomusBlobs riches its upper bound of performance faster. It is not the case for DataSteward which, thanks to its topology-aware distribution strategy of data nodes, manages the network better, resulting in a higher upper bound for I/O throughput. These observations demonstrate that our approach delivers high-performance being a viable option for Big Data applications.

Scenario 2: Memory usage for a scientific application. Many scientific applications have rigid resource constraints and their successful execution depends on meeting these criteria. An important feature brought to cloud applications by our proposal is a better usage of their VMs resources (CPU, memory and disks). The goal of this evaluation is to analyze to what extent does sharing the VMs for managing data affects the computation. To perform this analysis, we used a bio-informatics application which enables a configurable execution in terms of memory usage. The computation performs the Kabsch algorithm [105], which computes the optional rotation matrix that minimizes the root mean squared deviation between two sets of data. This algorithm is used in many scientific computations from fields like statistical analysis or cheminformatics for molecular structures comparison.

The experimental methodology consisted in running the application with increasing amounts of memory used for the actual computation. The remaining memory is assigned to the collocated storage, for which we used the TomusBlobs approach as before. The execu-

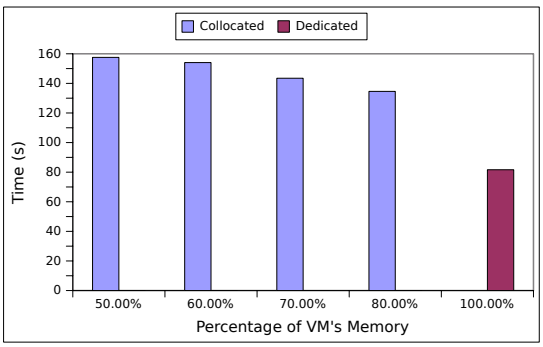


Figure 10.6: The execution time of the Kabsch-based application on a fixed set of input data. The percentage of the VM memory used by the application is increased, when using the collocated storage. The final bar represents the application execution time when all VM memory is used for the computation and the storage is moved to dedicated nodes

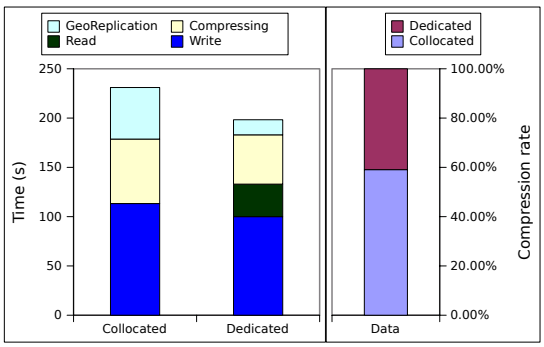


Figure 10.7: At left, the total time spent to compress and geographically replicate 100 MB of data from the Kabsch-based application, when these operations are performed on the application nodes and with DataSteward. On right, we show the gains in compression rate obtained when data is aggregated first from multiple nodes, before compressing it.

tion times of this scenario were compared with the case in which all the memory of the VMs is used for computation and the storage is handled by DataSteward, located on dedicated nodes. The results are shown in Figure 10.6. As for many similar applications, using more of the VMs resources can reduce the computation timespan (see the different times when using the collocated storage). Nevertheless, performing auxiliary tasks to support the main computation, such as handling data, also requires a share of the compute infrastructure. Our dedicated node approach solves these contradicting constraints. The application can harness all its local computing resources while the data is available at a high access rate within the deployment. Therefore, DataSteward reduces the execution timespan (computation and data handling time) for such scenarios to half compared to a collocation strategy.

10.3.3 Data Processing Services for a Scientific Application

Finally, we evaluate the 3 data processing services provided by the DataSteward: data compression, geographical replication and the toolkit for file group operations. For this analysis we consider two sets of data. First we use the data computed by the Kabsch-based application, having a size of approximately 100 MB, which is compressed and geographically replicated from the European to the United States data center. The second data set is the 28,000 log files, each file having a size less than 10 KB, resulted from the A-Brain experiment described in Chapter 9. The most common operation on these files, performed during the long-running experiment, are used to assert the benefits brought by the toolkit for file group operations of DataSteward.

We start the evaluation with the data compression and geographical replication services offered by DataSteward. We compare them with the default option for obtaining these functionality, in which each users implements the corresponding logic on the application nodes (i.e., collocation), therefore adding an extra-overhead to the main computation. The execution times are depicted in Figure 10.7. The evaluation scenario considers the total times,

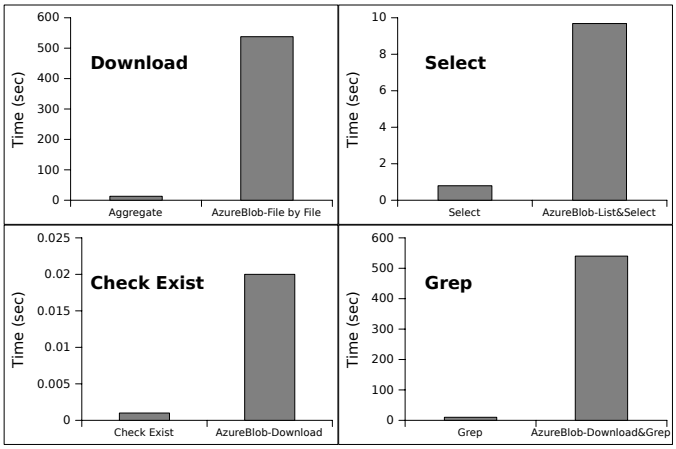


Figure 10.8: The timespan of executing recurrent operations from scientific experiments on 28000 monitor files, when the operations are supported and executed by the storage service or implemented at client side.

shown in left of Figure 10.7, to write the data to the storage system, compress it and transfer it to the remote data centers. Additionally, we present in right side of Figure 10.7 the levels of compression achieved when the operation is applied independently on each file, as in the case where the functionality is provided on each node (i.e., collocated), or collectively on the entire data set (i.e., dedicated). The price paid by DataSteward to execute these operations non-intrusively is an extra transfer from the application nodes to the dedicates storage node. Such a transfer is not required when each node handles its own data locally, therefore the missing "read" label for the collocated option in Figure 10.7. Nevertheless, the overall execution time is reduced by 15 % with DataSteward. This is because it is more efficient to transfer locally and then compress all the aggregated data at one place than to do it independently on small chunks in the application nodes. Building on this grouping of data, DataSteward is able also to obtain up to 20 % higher compression rates. This is a representative result for other scientific applications, showing that our solution can exploit better the similarities (e.g., spatial, temporal) between the partial results computed per partitioned domain.

Handling large number of files. Big Data experiments generate and handle a large number of files: input, output, intermediate, log or monitoring data. Dealing with many files, either directly, by users, or autonomously, by the system, can be complex and takes time, especially if the necessary functionality is missing. To this purpose, we evaluate the efficiency with which DataSteward is able to perform operations on large groups of files as opposed to implementing these operations on top of the cloud storage. Figure 10.8 presents the execution times for these two options, for several file operations commonly observed during long-running experiments. As a data set, we use the 28000 monitoring files of the map tasks, from the A-Brain experiment. It can be observed that not only the cloud storage does not provide such functionality, but it also fails to support it efficiently for the applications compute nodes which implement it. DataSteward solves this issue by providing, through an extended API, file group operations at server side (i.e., in the compute nodes that hold the data). As a result, the time to manage the files, regardless the performed operation, is reduced with one order of magnitude. Such functionality eases and shortens the job of the scientists to prepare, deploy and manage their experimental environment in the clouds.

10.3.4 Going Further

The motivation to use a dedicated infrastructure inside the deployment was to enable a set of *data services* that deliver the power and versatility of the cloud to users' applications. The idea is to *exploit the compute capabilities* of the storage nodes to deploy data specific services to extend the current basic support for managing data. Such services cannot be run otherwise on the application nodes or, executing them in collocation would harness application resources and impact on the overall performance. DataSteward provides three advanced data services, which provide substantial benefits for scientific processing. In this section, we briefly introduce several other services that can leverage for Big Data applications the dedicated storage infrastructure.

Cache for the persistent storage. Its role would be to periodically backup into the cloud persistent storage the data from the dedicated storage nodes. The approach extends the hierarchical memory design principle introduced for TomusBlobs in Section 7.4 into an autonomous service, transparent to applications. As a result, the DataSteward storage alongside with all the provided services would be complemented with persistence capabilities, following closely the structure of the physical storage hierarchy: machine memory, local and network disks, persistent storage. For critical systems, this service can be coupled with the (already provided) geographically replication one. As a result, data can be backed-up across geographical-distinct cloud storage instances, guaranteeing data availability against disasters or data center outages. The client applications would access any storage only through the dedicated nodes. If the data is not available within the deployment (e.g., in case of a crash that affects all replicas of a data object), then the object is copied from the persistent storage, cached locally and made available to applications. Furthermore, different caching policies can be provided to accommodate specific needs and behavior of applications.

Cloud introspection as a service. The cloud model, as it is defined today, hides from users applications all infrastructure aspects: load, topology, connection routes, performance metrics, etc. On the one hand, this simplifies the task of building and deploying a cloud application, but on the other hand it prevents applications to optimize the usage of the leased resources. The proposed selection strategy demonstrates the benefits that topology information can bring for applications. Therefore, building on the clustering scheme presented in Section 10.2, one can design *an introspection service* that could reveal information about the cloud internals. Such a service would not contradict the current cloud model nor it would complicate in any way the development and the administration of applications, but would only provide extra knowledge for the interested parties. In a geographically distributed setting, this could result in applications having an intuition of the number of data centers, their location (i.e., latency) or interconnecting links. Within a data center, they can learn the type of topology used, the available bandwidth within and across the deployment or the number of physical machines and racks. The ability provided by our approach to collect large numbers of latency, bandwidth and throughput estimates without actively transferring data, provides applications an inexpensive way to infer the cloud's internal details. These hints can be used for topology-aware scheduling or for optimizing large data transfers. We further develop on this direction in the next chapters.

Geographically distributed data management. Supporting the management of data across geographically distributed sites is critical for Big Data applications as well as for federated clouds ("sky computing" [107]), as discussed in Section 9.3. DataSteward can address this issue by hosting a data transfer system on the dedicated nodes, while remaining non-intrusive to applications. This approach will mitigate the large-scale end-to-end data movements from the application to the data management service. Additionally, the compute capabilities, harness by DataSteward from the underlying compute nodes, can overcome the existing low throughput over wide-area networks which interconnects the cloud data centers. The service will act as an intelligent proxy for applications, by efficiently utilizing multicast schemes and effectively scheduling data transfer tasks in order to minimize data movements across data centers. This idea is developed and discussed in the next chapters, where we address the high-performance aspects of inter-data center communication.

Clearly, this list is not exhaustive. Other similar services can be defined by users and deployed on the dedicated nodes, following the extensibility principle presented in Section 7.4. The goal is to provide a software stack on top of the storage nodes, following a *data processing-as-a-service* paradigm. This "data steward" will be able, on the one hand, to optimize the storage management and the end-to-end performance for a diverse set of data-intensive applications, and on the other hand, to prepare raw data issued/needed by experiments into a science-ready form used by scientists.

10.4 Discussion

Many scientific applications are required to manage data in the VMs, in order to achieve new functionality or a higher throughput while reducing costs. However, doing so, raises issues regarding intrusiveness and reliability. To address such issues, we propose DataSteward, a data management system that provides a higher degree of reliability while remaining non-intrusive through the use of dedicated nodes. DataSteward aggregates the storage space of a set of dedicated nodes, selected based on a topology-aware clustering algorithm that we designed. The computation capabilities are capitalized through a data management layer which provides a set of scientific data processing services that can overlap with the running applications. We demonstrated the benefits of this approach through extensive experiments performed on the Azure cloud. Compared to state of the art node selection algorithms, we show up to 20 % higher throughput. Compared to collocating data and computation strategy, our dedicated node approach reduces the execution times of scientific applications to half.

Encouraged by these results, next, we extend the service layer of DataSteward with additional data services that can facilitate Big Data processing on the clouds. In particular, we are interested to complement the user perspective with other topology-related information for enabling environment-aware optimizations and scheduling. Moreover, we focus on how we can capitalize on this compute capacity available to DataSteward, in order to offer high-performance transfers across data centers at configurable costs. The goal is to extend DataSteward as an enriched data management service for Big Data applications. These aspects are addressed in the next chapters of this thesis.

Chapter 11

Bridging Data in the Clouds

Contents

11.1 An Environment-Aware Approach for Inter-Site Transfers 123
11.2 The Cost of Performance across Data Centers 127
11.3 Validation and Experimental Evaluation 132
11.4 Discussion 140

This chapter develops the contributions published in the following paper:

- *Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers.* Radu Tudoran, Alexandru Costan, Rui Wang, Luc Bougé, Gabriel Antoniu. In Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2014), Chicago, IL, US, May 2014, pp. 92-101

Scientific applications are being ported on clouds to leverage the inherent elasticity and scalability of these emerging infrastructures. From large-scale scientific experiments (e.g., for climate modeling, genetics, high-energy physics) to global commercial online applications (e.g., office tools, data warehouse backup mechanisms, search engines, recommendation systems), all share a common feature. They produce and handle large data sets, in a highly geographically distributed manner. To enable such Big Data processing, cloud providers have set up multiple data centers at different geographical locations around the Globe. In such a setting, sharing, disseminating and analyzing the data sets results in frequent large-scale data movements across widely distributed sites. However, the existing cloud data management services lack mechanisms for dynamically coordinating transfers among different data centers with reasonable QoS levels and suitable cost-performance trade-off. Being able to effectively use the underlying compute, storage and network resources has thus become

critical for wide-area data movements as well as for federated cloud settings; as shown also by the A-Brain Big Data experiment, discussed in Chapter 9.

The handiest option to handle data distributed across multiple data centers is to rely on the cloud storage service. This approach allows to transfer data between arbitrary endpoints by intermediately storing it. Several systems adopt this approach to manage data movements over wide-are networks [112, 129]. Such services focus on data storage primarily and support other functionality essentially as a "side effect". Typically, they are not concerned by achieving high throughput nor by potential optimizations or differentiated QoS, e.g., cost/performance trade-off. A number of alternative solutions emerged in the context of the *GridFTP* [3] transfer tool, initially developed for grids and then adapted to clouds. Among these, the most representative is GlobusOnline [4], which provides file transfers through intuitive Web 2.0 interfaces. However, Globus Online only performs transfers between GridFTP instances. It remains unaware of the environment and therefore its transfer optimizations are mostly done statically. Moreover, the framework focuses on administrating and managing warehouse/grid files rather than interconnecting application instances running across data centers. Alternative approaches (some being extensions appeared in the context of aforementioned GridFTP) build on the *network parallelism* to increase the transfer performance, using multi-hop path splitting [109, 118] or multiple paths [148] to replace direct TCP connections. Others transfer mechanisms exploit the *end-system parallelism* through parallel streams [80] or concurrent transfers [119].These solutions come at some costs: under heavy load, per-packet latency may increase due to timeouts while extra memory is needed for the receiving buffers. Additionally, these techniques cannot be ported to the clouds, since they rely strongly on the underlying network topology, inaccessible at the user-level. Moreover, the parameters which these systems tune statically do not consider the important cloud aspects such as costs, interconnecting routes between VMs or inter-site latencies.

Our approach in a nutshell. To address the problem of data dissemination across geographically distributed sites we introduce a cloud-based data transfer system which dynamically exploits the network parallelism of clouds via multi-route transfers, offering predictable cost and time performance. The problem of low and unstable interconnecting throughput between data centers is addressed through enhanced data management capabilities which adopt in-site replication for faster data dissemination according to cost-performance constraints. The nodes involved in the replication scheme are used as intermediate hops to forward data towards destination. In this way, extra bandwidth capacity is aggregated and the cumulative throughput of the system is increased. Moreover, our system automatically builds performance models for the cloud infrastructure, in order to efficiently schedule the data transfers across the intermediate nodes and effectively utilize these resources. To this purpose, the system monitors the environment, and estimates dynamically the performance models for cloud resources to reflect the changing workloads or varying network-device conditions. The key idea is to predict I/O and transfer performance in order to judiciously decide how to perform transfer optimizations over federated data centers. In terms of efficiency and usage, the approach provides the applications with the possibility to select a trade-off between cost and execution time. The transfer strategy is then optimized according to the trade-off. The results show that the approach is able to reduce the financial costs and transfer time by up to 3 times.

11.1 An Environment-Aware Approach for Inter-Site Transfers

Application instances running across multiple data centers require efficient tools for communication and data exchanges. In addition to providing such functionality, one needs to consider integrating it into a uniform data management system alongside with the storage and other data services. Otherwise, dealing with multiple data-specific APIs can make the task of migrating and executing scientific applications on the clouds complex. Therefore, we extend the DataSteward storage approach, discussed in Chapter 10, with a high-performance transfer system. Moreover, the solution builds on the idea of an introspection service for the cloud environment by monitoring, modeling and predicting resources performance. The approach is designed for any application as it runs in user-space without requiring modifications to the cloud middleware nor elevated privileges. In the process of designing the solution, we extended the list of design principles introduced for TomusBlobs in Chapter 7.1 and DataSteward in Chapter 10.1 with 3 additional ones.

Environment awareness. Cloud infrastructures are built on commodity hardware and exploited using a multi-tenancy model. This leads to variations in the delivered performance of the nodes and the communication links. Monitoring and detecting in real-time such changes is a critical requirement for scientific applications which need predictable performance. The monitoring information can then be fed into higher-level management tools for advanced provisioning and transfer scheduling purposes over wide-area networks. This helps removing the performance bottlenecks one by one and increases the end-to-end data transfer throughput. Furthermore, this information about the environment is also made available to applications as a stand-alone service, i.e., *introspection as a service*.

Modeling cloud performance. The complexity of data center architectures, topologies and network infrastructures make simplistic approaches for dealing with data transfers less appealing. Simply exploiting system parallelism will most often incur high costs without reliable guarantees about the delivered performance. Such techniques are at odds with the typical user goal of reducing costs through efficient resource utilization. Therefore, accurate performance models are needed, leveraging the online observations of the cloud behavior. Our goal is to monitor the virtualized infrastructure and the underlying network and to predict performance metrics (e.g., transfer time or costs estimations). The main idea is to build these performance models dynamically and use them to optimize data-related scheduling tasks. As such, we argue for an approach that on the one hand provides enough accuracy for automating self-optimization of the data management; and on the other hand, remains simple and thus applicable in any scenario.

Cost effectiveness. As one would expect, cost closely follows performance. Different transfer plans for the same data may result in significantly different costs. Additionally, simply scaling the resources allocated for a task might remain inefficient as long as potential performance bottlenecks are not taken into account. In turn, this leads to unjustifiable expenses and resource wasting. Hence, a question that can be raised is: given the cloud interconnecting offering, *how can an application use it in a way that meets the right balance between cost and performance?* Our approach addresses this challenge

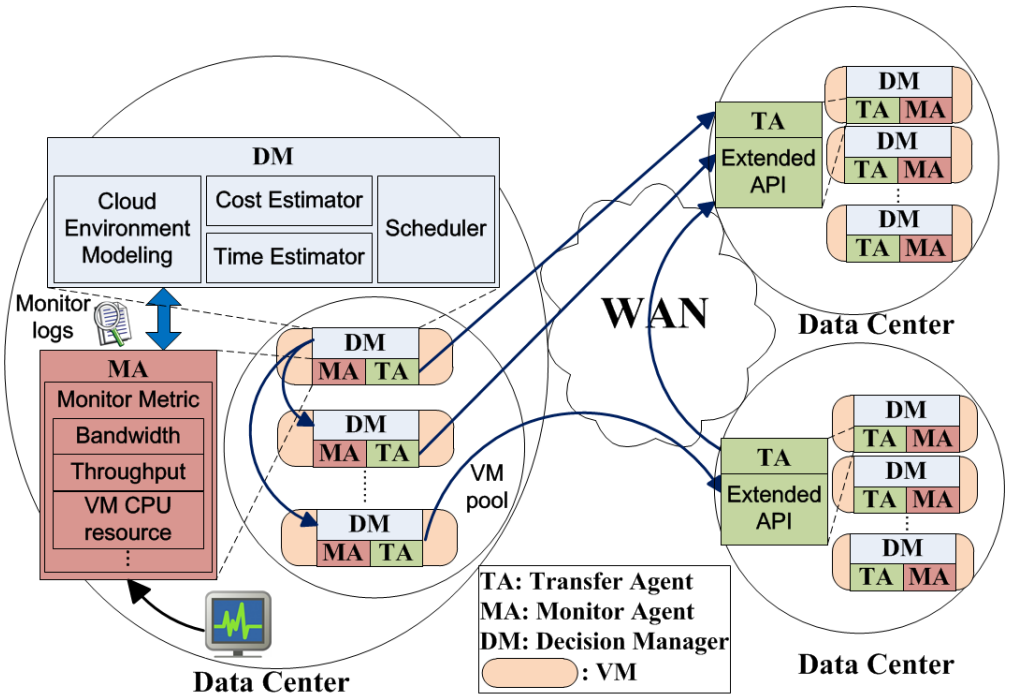


Figure 11.1: Architecture of the geographical distributed data management system.

by matching the resources costs with their delivered performance. As a result, applications are provided with the option of setting the desired trade-off between these aspects, choosing how much they spend to reach a certain performance level.

Our system relies on three components, refereed to as agents. These agents provide the following services: monitoring, data transfers and decision management. Agents are replicated on the VMs in the data centers where the application instances are running. The global architecture, denoted GEO-DMS, is depicted in Figure 11.1. The system is self-configurable, i.e., the instance within each VM discovers its peers automatically using the user credentials to query the cloud middleware about other deployments of the application from different sites (we recall that a deployment, described in Section 3.2, is limited to a single site). The resulting interconnected network is then used by each instance of the service to aggregate bandwidth by routing data packets across multiple data centers.

Applications simply interact with the local data management service to perform wide-area transfers using its extended API, complying with the principles of the DataSteward approach. Moreover, for easily integrating the approach with the other data services described in Chapter 10, the prototype was implemented in C# and validated on the Microsoft Azure cloud. The backbone of the system is a layered class hierarchy with approximately 50 classes, structured in three main modules corresponding to the conceptual components. This hierarchy is organized in a mostly single-rooted class library (i.e., the grouped classes inherit from a common class) following the abstract factory design pattern. Next, we present these 3 architectural entities and their functionality.

The Monitoring Agent (MA) has the role to monitor the cloud environment and report the measurements to a decision manager or to applications (i.e., providing introspection as

a service). Using the tracked data, a real-time online map of the cloud network and resource status is constructed and continuously updated. The metrics considered are: available point to point bandwidth, I/O throughput, CPU load and memory status. Additionally, cloud specific parameters, such as inter-site links and available routes across VMs are also accounted for, unlike any of the related works [80, 109, 118, 148]. New metrics can be easily defined and integrated using pluggable monitoring modules (see the extensibility principle of TomusBlobs in Section 7.4). The Monitoring Agent has also the role to record the monitoring history. Such a feature is important from two perspectives: on the one hand, the tracked logs are used by the scientists to understand and profile their cloud application executions better, and on the other hand, it provides the base functionality for a self-healing system.

The Monitoring Agent is designed as an ensemble of autonomous multi-threaded, self-configured subsystems which are registered as dynamic services, and are able to collaborate and cooperate in performing a wide range of information gathering tasks. In order to minimize intrusiveness on host systems, a dynamic pool of threads is created, and the threads are reused when a task assigned to a thread is completed. We have also set a customizable intrusiveness threshold, which limits the monitoring sampling frequency. This option is used for example by the decision manager to suspend the throughput measurements during transfers, as this information can be collected on the fly. The Monitoring Agent implements the monitored metrics as follows: the available bandwidth between the nodes and between the data centers is gathered using the `iperf` software [99]; the throughput performance and availability of interconnecting VM routes are computed by timing random data transfers; CPU and memory performance is evaluated based on a small benchmark, that we have implemented based on the Kabsch algorithm presented in Section 10.3.2.

The Transfer Agent (TA) performs the data transfers and exposes a set of functions used to exploit the network parallelism (e.g., `direct_send`, `forward_sending`, `read`, `split`, etc.) These functions are primarily used by the Decision Manager internally, to coordinate data movements across multiple routes over VMs and data centers. Data packets will be routed from one transfer agent to another, being forwarded towards the destination. In this way, the transfer agents exploit multiple paths aggregating the interconnecting bandwidth between data centers. Additional transfer optimizations include: data fragmentation and recomposition using chunks of variable sizes, hashing, acknowledgment for avoiding data losses and duplications of packages. One might remark that the at application level acknowledgment-based mechanism is redundant, as similar functionality is provided by the underlying TCP protocol. We argue that this can be used efficiently to handle and recover from possible cloud nodes failures, when intermediate nodes are used for transfers. For example, a packet which does not arrive at destination due to a failure of an intermediate node is resent by the sender through a different path. This action is generated after the expiration of the timeout period to receive the acknowledgment, or when an explicit request from receiver is received that marks missing packets in the data sequence. The mechanism is also used to monitor the ongoing transfers and to provide live information about the achieved throughput and the progress to completion by providing feedback from receiver to sender about transfer stats.

The Transfer Agent implements the data movements using parallel TCP streams. The parallelization of the transfer is done at the *cloud node level*: data is not entirely sent directly to the destination node, but parts of it are sent to *intermediate nodes*, i.e., other transfer agents from nodes located both in the same site and in other geographically distinct sites. This

approach leverages the empirical observation that intra-site transfers are at least 10x faster than the wide-area transfers and that the cloud architecture facilitate the routing of the packets through different switches, racks and network links. The intermediate nodes are then forwarding the data towards the destination, exploiting the multiple *parallel paths existing between data centers*. Data is sent as chunks extended with metadata information. Metadata is used for implementing the aforementioned mention optimizations: hashing, deduplication, routing or requests for acknowledgment stats. Additionally, it is used for recomposing the data at destination, as packets can arrive in any order depending on the distinct physical paths taken. The data transfer functions are made available via the extended API service introduced by DataSteward, using a set of simple commands similar to the FTP ones. Hence, the Transfer Agents can be controlled and used both locally as well as from remote locations, e.g., from other nodes, data centers, or from users on premise machines. The motivation of making all this functionality openly available is to provide users and workflows with the possibility to integrate data management into their own operations logic.

The Decision Manager (DM) coordinates the transfer from the source(s) to the destination(s), orchestrating the data flow, implemented by Transfer Agents, either through direct paths between sites or using also intermediate data centers. The application specifies the transfer parameters at the start of the transfer, providing the destination and either a preference trade-off between completion time and cost or absolute values for either of them. Based on these parameters and on the cloud status, the decision manager selects the appropriate number of resources to perform the transfer, so that it meets the requested efficiency constrains. By considering the global infrastructure (i.e., the data centers available for transfers), the manager decides whether the transfer is done directly between the nodes from the source and destination data centers, or using additional sites as intermediate hops. This selection is updated at specific intervals based on the cloud estimated performance, in order to reflect the inherent cloud resource variability. The decision manager considers the network of Transfer Agents to be similar to a global peer-to-peer network, on top of which it schedules and coordinates the multi-path transfers towards the destination. Although an instance of the Decision Manager is deployed on each nodes, for availability, each transfer is handled by a single entity, typically the one contacted by the application to initialize the transfer.

The Decision Manager schedules transfers using a set of modeling and prediction components that it implements. Based on them, the efficiency of the transfer is predicted at the beginning at the transfer, and then iteratively updated as the transfer progresses. The decision manager selects the optimal paths to be used. Then, it selects which nodes and resources to be provisioned, based on the time/cost prediction models, in order to comply with the transfer constraints requested by the application. For instance, it computes whether any economical benefit is brought by a set of resources or if the transfer cost is within the limit. Additionally, it checks if by using an increased number of nodes for parallel streaming of data, the completion time can be significantly reduced. The complete scheduling algorithm and the prediction models are presented in the next section. Moreover, the Decision Manager is also in charge of setting and adjusting the chunk sizes sent across each intermediate nodes, in agreement with the Transfer Agents, in order to maximize the resource usage and preserve the non-intrusiveness level within the imposed user limit. Finally, it has the role to supervise that the QoS performance obtained from the leased nodes meets the SLA advertised by the cloud. To this end, it detects any drops in resource performance and ei-

ther replaces them from specific tasks or/and interacts with the cloud middleware to change them from the application deployment.

11.2 The Cost of Performance across Data Centers

Multiple factors like multi-tenancy, wide-area networks or commodity hardware contribute to performance variation in a cloud environment [59]. Therefore, estimating the time or cost to perform an operation is not straightforward. The solution we propose to deal with this challenge is to monitor the environment and model its performance, which is then mapped with the costs and completion times of data operations. There are two options for modeling such complex infrastructures. *Analytical models* predict the performance of the underlying resources using low-level details about their internals alongside with workload characterizations. Although less expensive and faster than empirical models, they rely on simplified assumptions and require complex computation and sufficient details (e.g., applications, workload, infrastructure etc.) for better modeling. Due to the limited knowledge available in user space, such an approach is not adequate for clouds. Moreover, this option is not generic enough as it depends on the workload. *Sampling methods* perform active measurements of the targeted resources and do not require understanding the increasingly-complex, and mostly hidden, cloud internals. Our technique falls in the empirical, sample-based category. In this section we describe it, and then show how it is used for to predict the efficiency of transfers and to schedule them based on cost/completion-time trade-off.

11.2.1 Cloud Data Transfer Model

Modeling the cloud performance in time, based on the monitored samples, implies some form of time series analysis. The approach consists in taking a (sub)set of the collected samples points and defining a parametrized relation between samples (or between samples and the samples to be integrated), which defines the model. This model is then used to perform predictions. Many works have been proposed in the area [26, 41, 166, 176], either focusing on optimizing the prediction accuracy for precise scenarios or proposing complex computational methodologies (e.g., clustering, classification, regressions, kernel-based etc.) to generally address the analysis. Applying such advanced time series techniques involves significant computations, which contradicts with our goal of providing a non intrusive solution. Additionally, it requires fine-grained refinements of the model's parameters, which is not easily done in the virtualized cloud environment nor always applicable without prior analysis. We take a different approach and design a simple technique, with low resource footprint, for integrating the newly collected monitored samples at runtime.

Our approach consists in making estimations about cloud performance, represented by any of the tracked metrics implemented by the Monitor Agent, based on information (i.e., samples) collected through monitoring. The model focuses on the delivered performance (i.e., what efficiency one should expect) and the instability of the environment (i.e., how likely is it to change). The main goal is to make accurate estimations but remain generic, such that the model is applicable regardless the monitored metric. Moreover, we account also for the potential cloud instability. To address this, we propose to weight the trust given to each sample dynamically, based on the environment behavior observed that far. The

resulting inferred trust level for the sample is then used to update the knowledge about the environment performance (i.e., to define the relation between samples).

The monitoring information used for performance estimation is collected in two phases: an initial learning phase, at deployment start-up (in case of DataSteward-based deployments this phase overlaps with the node selection process, described in Section 10.2); and a continuous monitoring phase during the lifetime of the deployment. The measurements are done at user configurable time intervals, in order to keep the system non-intrusive to the application. The model consists in defining the time series relation (for both cloud average performance and variability) between accumulated samples and the new samples. This is applied at each time moment i. We use a fixed number of previous samples, denoted h. Let S be the value of the monitored sample to be integrated in the model at that moment.

The proposed time series relations are shown in Equation 11.1 for the cloud average performance, denoted μ, and in Equation 11.2 for the corresponding variability, denoted σ. As it can be observed, we use a moving average between the history samples and the new sample (i.e., $\frac{(h-1)\times existing\_knowledge + new\_knowledge}{h}$). However, to account for the cloud variability, the new sample is weighted (w) and integrated proportionally with the current average. Thus, we compute the *new_knowledge* as the weighted mean between the sample and the previous average (i.e., $new\_knowledge = (1-w) * \mu_{i-1} + w * S$), thus obtaining Equation 11.1. For computing the variability between samples, one could rely on the default formula for standard variability, i.e., $\sigma_i = \sqrt{\frac{1}{h}\sum_{j=1}^{h}(x_j - \mu_i)^2}$. However, this would require to memorize all h previous samples. To save memory, we rewrite (i.e., raise to power and break the sum) the standard variability equation, at moment i, based on its previous state, at moment $i-1$. For modeling these states iteratively at successive time moment, we introduce the γ parameter in Equation 11.3, as an internal parameter, based on which we update the variability (σ). The resulting formulas are shown in Equations 11.2 for variability and Equation 11.3 for the internal parameter.

$$\mu_i = \frac{(h-1) * \mu_{i-1} + [(1-w) * \mu_{i-1} + w * S]}{h} \tag{11.1}$$

$$\sigma_i = \sqrt{\gamma_i - \mu_i^2} \tag{11.2}$$

$$\gamma_i = \frac{(h-1) \times \gamma_{i-1} + w \times \gamma_{i-1} + (1-w) \times S^2}{h} \tag{11.3}$$

As aforementioned, our approach consists in weighting each sample individually and integrating it proportionally. Properly and dynamically selecting the weight is important as it enables both to filter out temporal performance glitches and to react to actual performance changes. Therefore, to define such a weighting strategy we rely on the following principles.

- A high standard deviation will favor accepting any new samples (even the ones farther from the known average), as it indicates an environment likely to change.
- A sample far from the average is a potential outlier and it is weighted less.
- Less frequent samples are weighted higher, as they are rare and thus more valuable than frequent samples.

We synthesize these weighting strategy in Equation 11.4. We model the first 2 previous principles using the Gaussian distribution (i.e., $(e^{-\frac{(\mu-S)^2}{2\sigma^2}})$. The choice comes naturally as this distribution enables to compute the weight according to the observed variability and the distance of the sample from the known average. The third principle is modeled such that the time frequency of the samples (tf) within a time reference interval (T) gives the weight (i.e., $\frac{tf}{T}$. However, in order to adjust this model to weight higher the less frequent samples (e.g., having only 1 sample in the time reference T should be weighted more than having T samples — the maximum), we subtract their ratio from 1. Finally, we average these two components and obtain the normalized weight in Equation 11.4, with 0 meaning no trust and 1 indicating full trust for the sample. For weight values of 1, Equation 11.1, becomes the time series formula for moving average. This shows that our model regards all samples equally only in stable environments with samples distributed uniformly.

$$w = \frac{e^{-\frac{(\mu-S)^2}{2\sigma^2}} + (1 - \frac{tf}{T})}{2} \tag{11.4}$$

11.2.2 Efficiency in the Context of Data Management

Efficiency can have multiple meanings depending on the application context and the user requirements. The declinations we focus on, which are among the most critical in the context of clouds, are the transfer time and the monetary cost. Therefore, we extend and apply the proposed performance model to characterize these aspects for data transfers across cloud data centers.

The Transfer Time (Tt) is estimated considering the number of nodes (n) that are used to stream data in parallel. These are the nodes holding Transfer Agents, which are used by the Decision Manager to route data packets across multiple paths. The second parameter used in the estimation of the transfer time is the predicted transfer throughput (thr_{model}). This is obtained from the previously described performance model applied to the monitor information collected about the cloud links. Let $gain$ is the time reduction obtained due to the use of parallel transfers. This value is determined empirically and has values less than 1, i.e., 1 represents a perfect speedup. The speedups obtained with respect to the degree of parallelism (i.e., number of nodes used) are analyzed in Section 11.3.3. Equation 11.5 gives the transfer time by adjusting the default method of computing transfer time (i.e., $\frac{Size}{Throughput}$) with the performance gained due to parallel transfers (i.e., $\frac{1}{1+(n-1)\times gain}$). The formula for the reduction coefficient is obtained as follows. For non parallel transfers (i.e., $n = 1$) there is no time reduction, this coefficient is canceled, thus the $n - 1$ term. For actual parallel transfers, we associate the reduction $gain$ with the parallelism degree. Finally, we invert the coefficient function and make it decrease from 1 to 0 in order to give the time reduction (e.g., if we would model the throughput, this last stage would not be necessary as the throughput increases with the parallel transfers).

$$Tt = \frac{Size}{thr_{model}} \times \frac{1}{1 + (n-1) \times gain} \tag{11.5}$$

The Cost of a transfer across data centers is split into 3 components. The first part corresponds to the cost charged by the cloud provider for the amount (*Size*) of outbound data ($outbound_{Cost}$), as usually inbound data is free. If these would change in the future, our model can integrate the new reality by simply adding this cost component in a similar manner. The other two cost components represent the cost derived from leasing the VMs (with n being the number of VMs) which perform the transfer. These price components are the costs associated with the used network bandwidth (VMC_{Band}) and the CPU cycles used (VMC_{CPU}). However, these VMs are not always fully used for a transfer. Therefore, we assign a percentage for the usage level of these resources. This percentage is in fact the intrusiveness level ($Intr$), or the resource footprint, generated by the transfer. The cost of these components is thus computed by multiplying usage time (Tt) with the used resources and the cloud prices. The final cost is obtained by adding these components, as shown in Equation 11.6, which is rewritten to factor out common terms. For simplifying the model, we considered that each of the n nodes sends the same amount of data, resulting in $\frac{Size}{n}$ data sent per node.

$$Cost = outbound_{Cost} \times Size + n \times Intr \times Tt \times VMC_{CPU} + n \times Intr \times \frac{\frac{Size}{n}}{Tt} \times VMC_{Band} \tag{11.6}$$

In addition to the estimations about the transfer time and cost, this model offers another important befit.It captures the correlation between performance (time) and cost (money). This enables our system to adjust the trade-off between them dynamically during transfers. An example of such a trade-off is setting a maximum cost for a data transfer. Based on this budget, our solution is able to infer the amount of resources to use.Moreover, as the network or end-system performance can drop due to cloud resource performance variations, the system rapidly detects the new context and adapts to it in order to meet the budget constraints. Finally, our approach offers to applications an evaluation of the *returned benefit* from the leased resources. In turn, applications can leverage this knowledge to select their optimal expense.

11.2.3 Multiple Data Center Paths Transfer Strategy

Sending large amounts of data between two data centers can rapidly saturate the available interconnecting bandwidth. Moreover, as our empirical observations show, the direct connections between two data centers is not always the fastest one. This is due to the different ISP infrastructures which interconnect the data centers, as these networks are not the property of the cloud providers. Considering that applications run on multiple sites, an interesting option is to use these sites as intermediate hops between source and destination. Therefore the transfer approach that we propose relies on multiple hops and paths instead of using just the direct link between the source and destination sites.

A similar idea for maximizing the throughput was explored in the context of grids. Several solutions were proposed in conjunction with the GridFTP protocol [108, 109]. However, for these private grids infrastructures, information about the network bandwidth between nodes as well as the topology and the routing strategies are publicly available to any users. Using this knowledge, transfer strategies can be designed to maximize certain heuristics [109]; or the entire network of nodes across all sites can be viewed as a flow graph and

the transfer scheduling can be solved using flow-based graph algorithms [108]. Nevertheless, in the case of public clouds, information about the network topology is not available to users. Regarding flow-graph algorithms, the missing information about the topology, which can partially be replaced by monitoring, would require a prohibitive intrusiveness level. All interconnecting links between nodes would need to be constantly monitored. Worse, the throughput of a link between data centers varies with the number of nodes used (see Section 11.3.3). Therefore, the links have to be monitored also for their performance when different number of parallel streams are used. Gathering such information to apply a flow-graph solution to maximize throughput via parallel transfers is clearly not efficient.

Selecting the transfer paths. We take a different approach to solve the paths selection problem across cloud data centers. The key idea is to consider the cloud as a 2-layer graph. At the top layer, a vertex represents a data center. Considering the small number of data centers in a public cloud (i.e., less than 20), it is very quick to perform any computations on this graph, e.g., determine the shortest path or second shortest path. On the second layer, each vertex corresponds to a VM of the system within a data center. The number of such nodes depends on each application deployment and can be dynamically scaled, according to the elasticity principle of the clouds. These nodes are used for fast local replication of data with the purpose of transferring data in parallel streams, as detailed for the Transfer Agents. As such nodes are allocated for the transfer, they aggregates bandwidth. The relation between layers is that the aggregated bandwidth at bottom layer is used in the top layer for computing the inter-site paths, as follows. The best path, direct or across several sites, is found and selected. Then, its throughput is increased by adding resource on it. However, as the throughput gains brought by the new resources on a path decrease, we also consider switching to different paths. As a result, our approach finds the best transfer topology using multiple paths across nodes and data centers.

Algorithm 4 implements this approach. The first step is to select the shortest path (i.e., the one with the highest throughput) between the source and the destination data centers (Line 5). Then, building on the elasticity principle of the cloud, we try to add nodes to this path, within any of the data centers that belongs to this shortest path, in order to increase its cumulative throughput (Lines 10–15). More nodes add more bandwidth, translating into an increased throughput. However, as more nodes are successively added along the same path, the additional throughput brought by them will become smaller with each new added node (e.g., due to network interferences and bottlenecks). To address this issue, we consider also the next best path (computed in the Algorithm 4 at Lines 7–8). Having these two paths, we can compare at all times the gain of adding the node to the current shortest path versus adding a new path (Line 12 in Algorithm 4). Hence, nodes will be added to the shortest path until their gains become smaller than the gain brought by a new path. When this is the case, this new path is added to the transfer network. The trade-off between the cost and performance, discussed in Section 11.2.2, is controlled and set by the user through the *budget*. This specifies how much the user is willing to pay in order to achieve higher performance. Our solution increases the number of intermediate nodes in order to reduce the transfer time as long as the budget allows it. More precisely, the system selects using the cost/performance *Model* from Section 11.2.2 the largest number of nodes, which keeps the cost defined in Equation 11.6 smaller than the *budget* (Line 2 in Algorithm 4).

Algorithm 4 The multi-path selection across data centers

```
1:  procedure MULTIDATACENTERHOPSEND
2:      Nodes2Use = Model.GetNodes(budget)
3:      while SendData < TotalData do
4:          MonitorAgent.GetLinksEstimation();
5:          Path = ShortestPath(infrastructure)
6:          while UsedNodes < Nodes2Use do
7:              deployments.RemovePath(Path)
8:              NextPath = ShortestPath(deployments)
9:              UsedNodes+ = Path.NrOfNodes()
                                          ▷ // Get the datacenter with minimal throughput
10:             Node2Add = Path.GetMinThr()
11:             while UsedNodes < Nodes2Use  &
12: Node2Add.Thr >= NextPath.NormalizedThr  do
13:                 Path.UpdateLink(Node2Add)
14:                 Node2Add = Path.GetMinThr()
15:             end while
16:             TransferSchema.AddPath(Path)
17:             Path = NextPath
18:         end while
19:     end while
20: end procedure
```

11.3 Validation and Experimental Evaluation

This section presents the evaluation of the proposed GEO-DMS transfer service. As before, the evaluation is done on the Microsoft Azure cloud using synthetic benchmarks and the A-Brain application scenario. Considering that our approach is designed for high-performance, multi-path transfers across sites, the experimental setup consists of all Azure data centers from United States and Europe (North-Central, South, East, West US and North, West EU). We used both the Small (1 CPU, 1.75 GB Memory) and Medium (2 CPU, 3.5 GB Memory) VM instances with tens of such machines deployed in each data center, reaching a total of 120 nodes and 220 cores in the overall system. From the point of view of the network bandwidth, the Medium instances have a higher expected share of the 1 Gbps Ethernet network of the physical machine than the Small ones, offering twice as more of the physical node resources.For the A-Brain-related experiments, we use the Extra-Large (8 CPU, 14 GB and 800 Mbps) VM instances, to comply with its resource requirements.

11.3.1 Evaluation of the Performance Model

To adapt to context changes, we propose a global monitoring tool for asserting various cloud performance metrics. However, just collecting status information about the cloud environment is not enough. In order to capitalize this information, the samples need to be integrated within a model in order to estimate and predict the cloud performance. We start the experiments by evaluating the accuracy of our cloud performance model, discussed in Section 11.2.1. The goal is to evaluate the accuracy obtained by dynamically weighting the samples compared to default monitoring strategy or other simple time series analysis option,

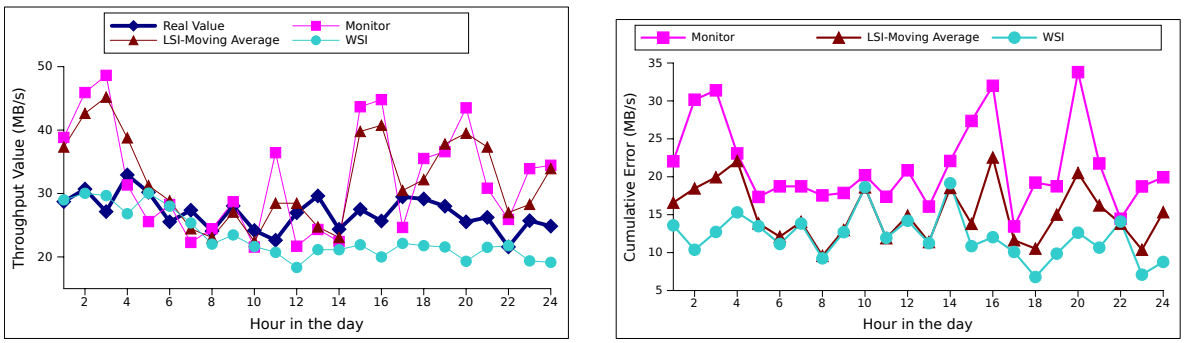


Figure 11.2: Left: Estimating the TCP throughput using multiple strategies for modeling the performance (Monitor and Update, Linear Sampling Integration and Weighted Sampling Integration — our approach). Right: we present the average aggregated error, per hour, corresponding to the TCP throughput approximation shown on the left.

as explained next.

We show in Figure 11.2 the evaluation for estimating the inter-site throughput in a 24-hour interval. Each sample shown in the figure represents the hourly averages computed based on 60 values. As previously stated, the most fundamental aspect about modeling the performance (i.e., time series analysis) is how to integrate (i.e., the relation) the new samples collected by the monitoring tools. We compared our weighted integration strategy (denoted *WSI*), with 2 other strategies. The first one (denoted *Monitor*) considers that the last monitored value represents from that point on the estimation of the performance, i.e, the last measurement is fully trusted and all knowledge about the environment is represented by it until the next sample is collected.Because of its simplicity and low cost (i.e., only one value needs to be retain per metric) it is currently the most used approach. The second strategy considers a linear integration of the samples (*LSI*), computing the estimation of performance as a linear average between the historical samples and the new sample. This option is the standard moving average strategy from time series analysis. It treats all samples equally, regardless the stability/instability of the environment or the frequency of the measurements.

On the left of Figure 11.2 we show how the actual throughput between North-Central US and North EU data centers is approximated using the 3 strategies for sample integration. On the right side, we report the difference (i.e., accuracy error) between the estimated values and the real values within the hourly interval. The default *Monitor* approach yields the worst results, being subject to all temporal performance variations. The linear integration strategy is also influenced by potential performance outliers collected by the monitor tool, but to a lesser extent. Our weighted approach has a smoother approximation of the actual throughput and is not as sensitive to temporary variations as the first option (or the second one for that matter). In fact, during periods with unstable performance (e.g., interval 1 to 5 or 18 to 24), dynamically weighting the samples based on the variability pays off. This approach enables to be more reactive to the samples, and to changes in the monitored performance, than using a fixed integration strategies where all samples are treated the same at any moment (LSI). When the performance variations are smaller, both the linear and the weighted strategies give similar results. This is explained by the fact that in a stable environment the weights our model assigns to the samples are close to 1 (see Equation 11.4). A weight value

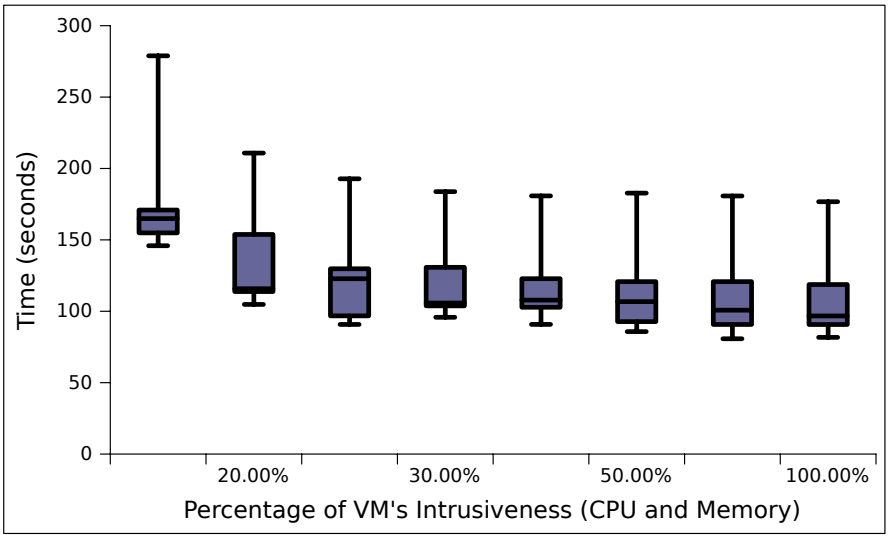


Figure 11.3: The impact of the intrusiveness on the transfer time of 1 GB of data between North-Central US and North EU Azure data centers. The number of VMs used is varied between 1 to 5, for each transfer measurement (i.e., for each segment) and represented as a whisker plot box.

of 1 is equivalent with the linear integration model, as the new samples are simply averaged with the history. This shows the strength and the reactiveness of our model which is able to adapt to the environment dynamics. Finally, the relative errors (10-15 %) of the model can be easily tolerated by the transfer service, when used for scheduling transfers, as they result in slightly moving the throughput performance around the maximum value.

11.3.2 Data Transfer Service Evaluation

We continue the experiments with the evaluation of the transfer times of our solution. The goal is to analyze the gains obtained by scheduling the transfers across multiple routes, in a cloud environment-aware manner, i.e., based on the cloud performance model. To this purpose, we first evaluate the impact of the intrusiveness level, captured in the model by the *Intr* parameters, on the transfer. Next, we compare the transfer time of our approach with state-of-the-art solutions for inter-site data transfers.

Evaluating the resource intrusiveness–transfer performance correlation. In general, the performance of a data transfer tends to be associated and explained only from the perspective of the communication link capacity (i.e., bandwidth). However, the amount of CPU and memory resources used by the nodes involved in the transfer, impact the performance as well. Our system is able to capture this and to operate within a defined intrusiveness level. Therefore, we analyze the impact of the resource utilization level on the wide-area transfers. Hence, we measure the performance when the intrusiveness level admitted is varied equally on all VMs involved in the transfer. The measurements consider the transfer time of 1 GB of data between the North-Central US to the North EU data centers. The number of nodes that are used for the multi-path transfer is varied from 1 to 5.

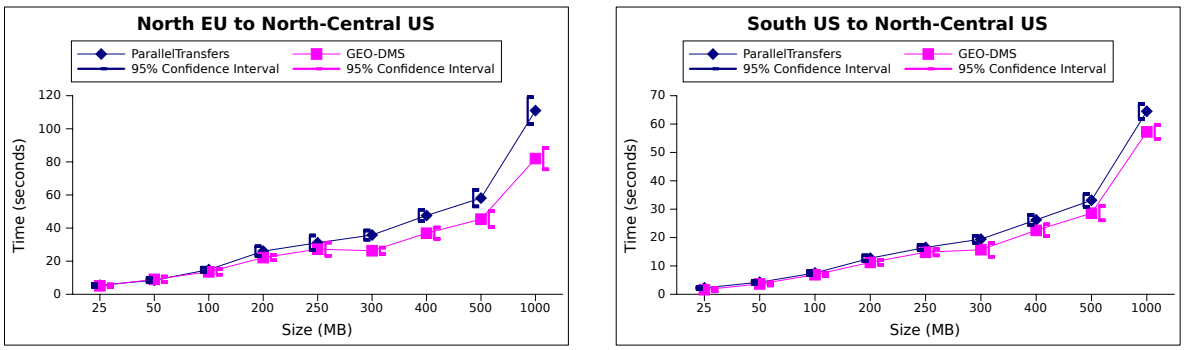


Figure 11.4: The transfer times of our approach (labeled as "GEO-DMS") compared to static parallel transfers. The results show the average and the 95 % confidence intervals for the transfer times for different data centers.

The results of these intrusiveness-performance measurements are shown on Figure 11.3. The highest values within each segment correspond to the situation when only 1 node is used for the transfer. The other information shown through the whisker boxes (e.g., minimum, and the lower, medium and upper quartiles) correspond to using multiple nodes. The transfer time reduction obtained for each extra node depends both on the intrusiveness level and on the cloud performance (analyzed in more details in the next experiments). The results show that adding more of the VM resources does not reduce the transfer time with the same percentage because: 1) the network bandwidth is bound, 2) replicating data within the site from the source to the intermediate nodes incurs an overhead, and 3) the VM resources performance vary. First, this observation shows that indeed, the amount of CPU or memory used impacts the transfer performance. Secondly, it shows that the effectiveness of resources to perform an operation is not constant. For example, having twice as much CPU costs 2 times more, but does not reduces the execution time with the same proportion. Hence, it is important to have a fine control of the amount of resources used, in order to increase the effectiveness of resource usage, decrease cost and tune performance. This is a strong argument which supports our choice for a data management system which allows applications or users to select the degree of resources utilization for each operation.

Evaluating the cloud environment-aware, wide-area parallel transfers. Using parallel streams to send data is a known technique to increase the transfer performance (e.g., increase throughput, decrease transfer time). However, there are no guarantees that simply sending data via several streams brings the maximum throughput out of the used nodes, let alone in the context data exchanges between cloud data centers. In fact, the following experiment, depicted on Figure 11.4, illustrates how the transfer efficiency of parallel transfers is improved using knowledge about the environment. We consider sending increasing data sizes from a source node to a destination between two closer (South and North US) and two farther (North EU and North US) data centers. The parallel transfer scheme is deployed on top of intermediate nodes located in the same data center as the source node. The experiment compares the performance obtained by such a parallel transfer (i.e., data is simply split and sent via the available streams across nodes) with our approach which integrates in the parallel transfer scheduling knowledge about the environment. The Decision Manager adapts to drops in CPU or bandwidth performance by discriminating the problematic nodes (i.e.,

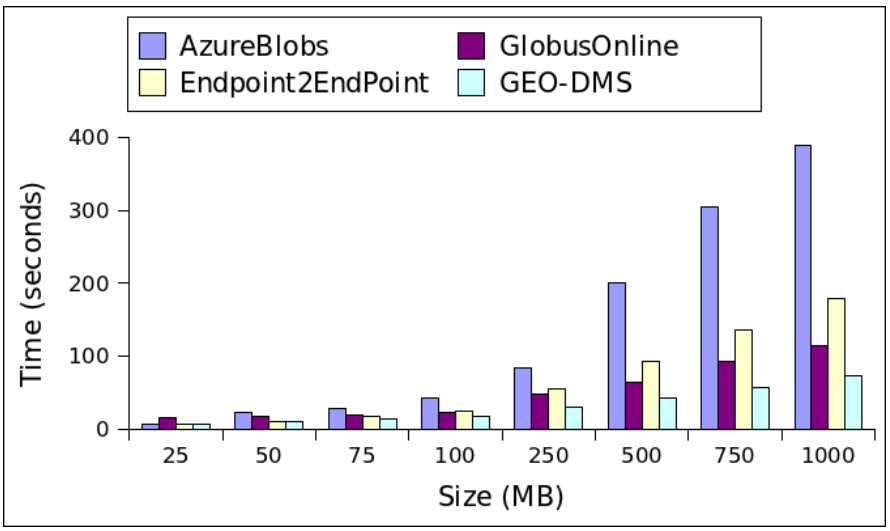


Figure 11.5: Comparing the transfer time of our approach with other existing options, for increasing volumes of data.

sending less data through those links) or by selecting some alternative nodes if available and better. Both the default parallel transfer and our approach (denoted "GEO-DMS") use equal amount of resources. The results on Figure 11.4 show that with the increase in size of data and in geographical distance between data centers, our approach reduces more the transfer times. Therefore, with longer transfers and potentially more cloud performance variability, being environmentally-aware pays off. This is an important observation in the context of Big Data applications, which are the ones typically deployed across sites and operating and transferring massive amounts of data. Considering also the 95 % confidence intervals, also shown in Figure 11.4, we can observe that our approach reduces the transfer time with more than 20 % over simple parallel transfers strategies.

Comparing the performance with the existing solutions. As previously stated, cloud lack support for inter-site application data transfers. However, several existing solutions can be used or built as a placeholder. Next, we compared our data transfer service with such state-of-the-art stubs: the Globus Online tool (which uses GridFTP as a server backend), Azure Blobs (used as an intermediate storage for moving data) and direct transfers between end-points, denoted EndPoint2EndPoint. The results of this evaluation are shown on Figure 11.5. Azure Blobs is the slowest option as the transfer is composed of a writing phase, with data being written by the source node to the storage, followed by a read phase, in which the data is read by the destination. These steps incur significant latencies due to the storing operation, the geographical distance (of the source or destination, depending on the location of the cloud storage instance with respect to them) and the HTTP-based access interfaces. Despite these performance issues, Azure Blobs-based transfers are currently the only cloud offering for wide-area data movements. Globus Online is a good alternative but it lacks the cloud-awareness. Moreover, it is more adequate for managing (e.g., list, create, view or access based on permission the files) grid/warehouse data rather than supporting application data exchanges. Finally, the EndPoint2EndPoint fails to aggregate extra bandwidth between data centers as it implements the transfers over the basic, but widely used, TCP client-server

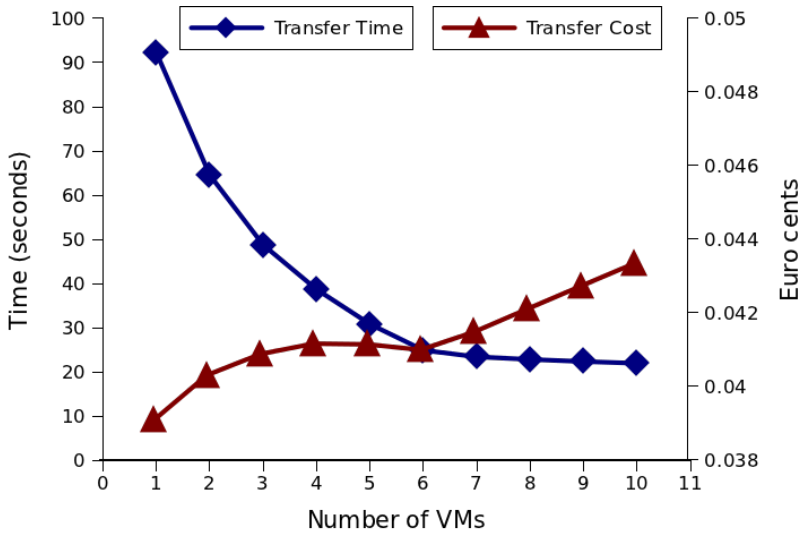


Figure 11.6: The trade-off between transfer time and cost, when using multiple VMs. The values are computed for to 1 GB transfers between North EU and North-Central US.

socket communication. Our solution, which leverages multi-route transfers for clouds, reduces the overall transfer time by a factor of 5 over the default cloud offering and by up to 50 % over the other transfer options that can be adapted for the cloud. Budget-wise, these approaches do not incur the same costs. Azure Blobs adds extra cost for using the persistent storage. EndPoint2EndPoint corresponds to the use of 1 node and the cost/performance trade-off is analyzed in the next section. For the comparison with Globus Online, the same setup was used, meaning that higher performance corresponds to lower costs. These results show that our data transfer service is indeed able to provide cloud applications with high-performance data transfers across multiple sites.

11.3.3 The Cost–Execution Time Efficiency of Data Transfers

The next set of experiments focus on the efficiency aspects of our approach. As users rent and pay for the resources, we are interested that the transfer service obtains the maximum benefit out of them. Therefore, we analyze the correlation between the delivered transfer performance level and the incurred costs and show how this benefits applications.

Evaluating the transfer efficiency. On Figure 11.6, we propose an experiment to evaluate the relation between the transfer performance and the monetary cost. We map the costs with the performance gains (i.e., time reduction) obtained by the parallel transfer implemented over intermediate nodes. These nodes have an associated cost, as users either lease and dedicate them (e.g., DataSteward) or collocate them with the main computation, sharing the nodes resources (e.g., TomusBlobs). As we scale the degree of parallelism, we observe a non-linear reduction for the transfer time. Each new node added for the transfer brings a lower performance gain. This decrease in performance per added node is due to the overhead of replicating data, within the site, from the source node to the intermediate nodes. At the same time, the cost of the nodes remains the same, regardless the efficiency of their usage. Such situations make the task of computing the cost of performance for cloud infrastructures

challenging.

The time reduction obtained by using more nodes prevents the transfer cost to grow significantly, up to a certain point. This can be observed on Figure 11.6 when using 3 up to 5 VMs. This happens because the cost is charged based on the period the nodes are used for the transfer (see Equation 11.6). Thus, despite paying for more resources, the transfer time reduction balances the cost. When scaling beyond this point, the performance gains brought by the new nodes, start to decrease. Hence, as the transfer time stabilizes (i.e., reaches the lower bound) the cost increases with the number of nodes. Looking at the cost/time ratio, an good point is found with 6 VMs for this case (a significant time reduction for a small extra cost). However, depending on how urgent the transfer is, different transfer times can be considered acceptable. In this logic, critical operations must be provided with the possibility to scale the transfers over multiple resources, and thus pay more, for reducing the times and meeting the deadlines. Hence, having such a mapping between cost and transfer performance, as shown in Figure 11.6 and modeled by Equation 11.5 and 11.6, enables our system to meet the critical challenge of enabling applications to customize their cost/time trade-off.

Evaluating the multiple-path transfer strategy across data centers. So far, the evaluation focused on sending data from the nodes in the source data center to the destination data center. Now, we consider the scenario in which the nodes from additional data centers are used as intermediate hops to transfer data. For this experiment, we use all the 6 US and EU Azure sites, our transfer service being deployed in all of them. Figure 11.7 presents the evaluation of our multi-path transfer approach across data centers, described in Algorithm 4. We compare our solution with 3 other transfer strategies which schedule the transfer across multiple nodes. The first option, denoted DirectLink, considers direct transfers between the nodes of the source data center and the destination. All nodes available within the budget constraints for the transfer are allocated in these two sites. As the direct link between source and destination might not be in fact the best one from the point of view of throughput of transfer time, the other strategies consider the best path (or "shortest path") computed using Dijkstra's algorithm. The best path can span over multiple data centers and thus the nodes allocated for transfers are scheduled across all of them. The selection of the best path for routing the transfer can be done: 1) once, at the beginning of the transfer (this option is denoted ShortestPath static); or 2) each time the monitoring systems provides a fresh view of the environment (denoted ShortestPath dynamic). In fact, the static strategy across the shortest path shows the throughput which can be obtained when the transfer is not coupled with the monitoring system. For all strategies, an equal number of nodes was used to perform the transfer.

On left of Figure 11.7, we present the accumulated throughput achieved when 25 nodes are used to perform data transfers between data centers. We notice that the performance of the shortest path strategy and the one that we propose are similar for the first part of the transfer. This happens because our scheduling algorithm extends the shortest path algorithm with mechanisms for selecting alternatives paths when the gain brought by a node along the initial path becomes smaller than switching to a new path. The improvement brought by this increases with time, reaching 20 % for the 10-minute window considered. Alternatively, selecting the route only once, and thus not being cloud environment-aware, decreases the performance in time, becoming inefficient for large transfers. On the right of Figure 11.7,

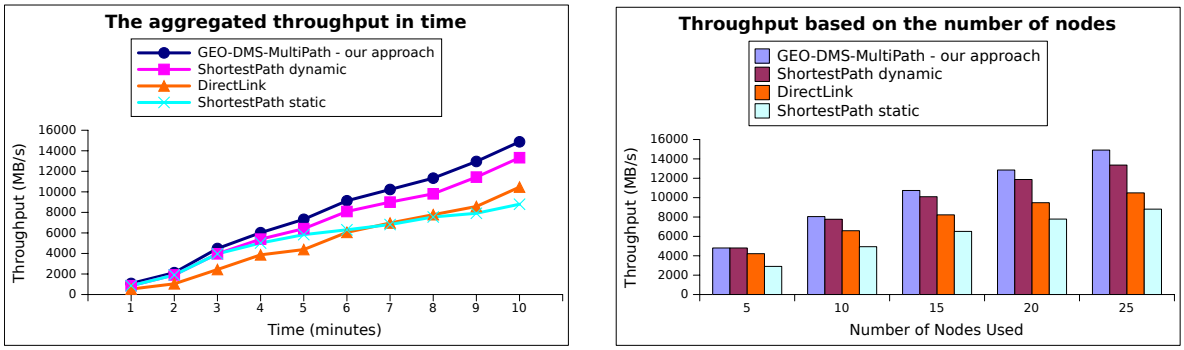


Figure 11.7: The throughput between NEU and NUS data centers for different transfer strategies across parallel paths over multiple sites. We consider all the 6 Azure data centers from US and EU. The throughput is evaluated: Left — with respect to timespan, while the overall number of nodes across all data centers is fixed to 25 ; Right — with respect to number of nodes, while the time frame in which data is sent is kept fixed to 10 minutes

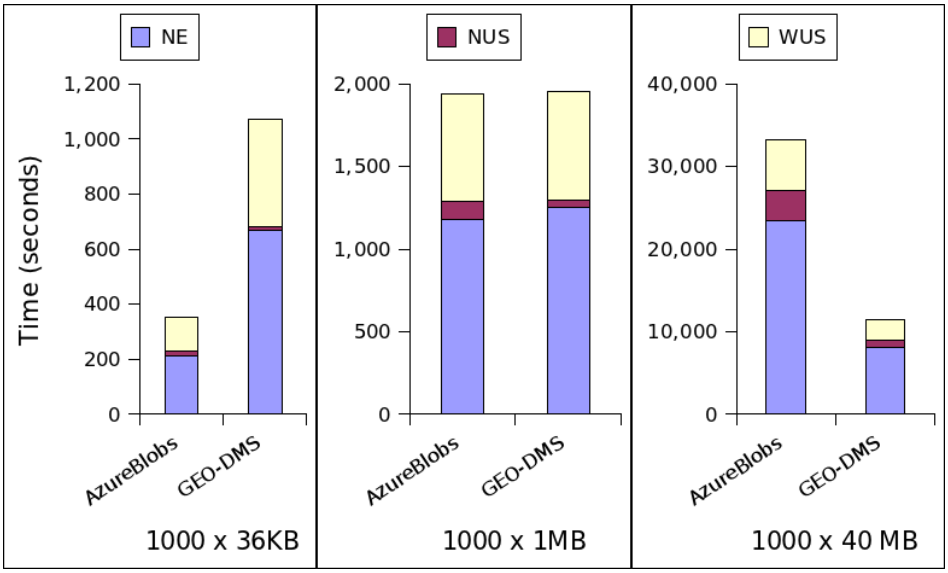


Figure 11.8: Execution times of the A-Brain application across 3 data centers, using Azure Blobs and the GEO-DMS transfer service. In each of the 3 experiments, the bars indicate the total time of transferring the partial result files from each data center (NE, NUS and WUS) towards the Meta-Reducer located in NUS. The time scale of the experiments differs.

we analyze the throughput levels obtained for increasing number of nodes. The goal is to analyze how much data can be sent given a number of nodes and a time-frame (here 10 minutes). We observe that for a small number of nodes, the differences between the strategies are very small. However, as more nodes are used for the transfer, distributed across different geographical sites, our algorithm is capable to orchestrate their placement better and achieve higher throughput. These results show that our approach is suitable for Big Data applications, being able to efficiently use the cloud resources for massive data movements.

Evaluating the benefits in a real-life scenario. Finally, we present an evaluation of the time reductions that can be obtained for wide-area transfers in the context of a real-life application. We use the large A-Brain computation scenario across 3 data centers, described in Section 8.3. We focus on the data movements required to compute the final global result. The comparison considers the transfer times of 1000 files, representing partial data, sent from each data center running the TomusBlobs MapReduce engine towards the Meta-Reducer. We compare the usage of the proposed transfer service (GEO-DMS) with transferring data using Azure Blobs, as it was initially the case for the multi-site hierarchical MapReduce (Chapter 8). The results are shown on Figure 11.8 for multiple file sizes, resulted from different input data sets and configurations. For small data sets (108 MB from $3 \times 1000 \times 36$KB files), the overhead introduced by our solution due to the extra acknowledgments, makes the transfer less efficient. However, as the data size grows (120 GB), the total transfer time is reduced by a factor of 3.This shows that our solution is able to sustain high-performance data management in the context of large-scale scenarios and Big Data applications.

11.4 Discussion

Many large-scale applications need tools to transfer data efficiently between their instances across data centers. Moreover, managing and customizing the cost for achieving the desired level of performance is critical. To this purpose, we introduce a cloud-based data management system for Big-Data applications running in large, federated and highly dynamic environments. Our solution is able to use effectively the interconnecting networks between the cloud data centers through an optimized protocol for scheduling the transfer resources, while remaining non-intrusive and easy to deploy. At its core, it uses a sampling-based model for cost-performance in a cloud setting to enable efficient transfer operations across a group of geographically distributed data centers. As an example, by distributing data locally, it enables high wide-area data throughput when the network core is underutilized, at minimal cost. Our experiments show that the system achieves high-performance in a variety of settings. It substantially improves throughput and reduces the execution time for real applications by up to 3 times compared to state-of-the-art solutions.

Considering the high efficiency levels for managing the data across sites brought by this solution, we further explore its applicability from two perspectives. First, we are interested to apply the concept of multi-path transfers across cloud nodes for real-time data (i.e., streaming data across cloud data centers). A significant part of tomorrow's Big Data is expected to be produced as streams of events [23, 43, 172]. Therefore, extending the transfer service to support data streaming across cloud data centers is critical. We present this extension in Chapter 12. Second, we investigate this concept of transfer service from the perspective of the cloud providers. We believe that cloud providers could leverage this tool as a *Transfer as a Service*, which would enable multiple QoS transfer levels, cost optimization and significant energy savings. This approach and how it can be leveraged for Big Data applications is detailed in Chapter 13.

Chapter 12

Real-Time Data Management across Data Centers

Contents

12.1 Evaluating Strategies for Cloud Stream Processing 143
12.2 Modeling the Streaming of Data in the Context of Clouds 147
12.3 JetStream: Enabling High-Performance Streaming between Data Centers 151
12.4 Validation and Experimental Evaluation 154
12.5 Discussion 160

This chapter develops the contributions published in the following papers:

- *Evaluating Streaming Strategies for Event Processing across Infrastructure Clouds.* Radu Tudoran, Kate Keahey, Pierre Riteau, Sergey Panitkin and Gabriel Antoniu. In Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID 2014), Chicago, IL, US, May 2014, pp. 151-159
- *JetStream: Enabling High-Performance Event Streaming Across Cloud Datacenters.* Radu Tudoran, Olivier Nano, Ivo Santos, Alexandru Costan, Hakan Soncu, Luc Bougé, and Gabriel Antoniu.In Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems (DEBS 2014), Mumbai, India, May 2014, pp 23–34

The performance trade-offs inherent in current virtualization technology mean that data-intensive applications are often not the best fit for clouds and consequently cannot leverage the advantages of cloud computing [67, 103, 150]. This technological shortcoming is particularly impacting in the Big Data era. An increasing number of Big Data applications conform

to a pattern in which data processing relies on streaming data to a compute platform, located in a remote geographical location, where a set of similar operations is repeatedly applied. The data stream is composed of independent, but similar, units of data, called events, typically aggregating some states at a particular moment (e.g., the sensor readings at a certain moment). This compute paradigm is called stream processing.

The geographically distributed stream processing pattern is evident in scientific computation like virtual observatories such as the Ocean Observatory Initiative [137], with ocean sensors data being transferred in real time to cloud nodes for processing. Other stream processing scenarios appear when new acquired data is evaluated against the already known features (e.g., FluMapper [141]) or when the experimental data is processed as a series of time events [21, 102]. Geographical-distributed streaming is present also in commercial scenarios. Large-scale web services (e.g., Microsoft's Bing, Office 365, Google Docs) operate on tens of data centers around the globe. Typical operations such as continuously aggregating monitoring data from all service instances, assess in real-time the QoS of the global service or running global data mining queries, all require to gather in real-time information from multiple sites, i.e., multi-site stream processing. Hence, enabling fast data transfers across the geographically distributed sites becomes particularly important for such applications, due to the real-time nature of their computation. Consequently, such scenarios require high-performance real-time communication in order to adapt fast to any changes, scale rapidly to fluctuating requests and compute the answer within a specified delay.

Strategies for evaluating the cloud support for streaming. In order to identify and understand the challenges brought by stream processing across cloud sites, we start with a performance evaluation study. To this purpose, we propose and analyze two strategies for implementing inter-site cloud streaming, i.e., transferring data to a geographical-remote site for processing. The first strategy seeks to overlap computation and communication by streaming data directly to the nodes where the computation takes place, in such a way that the rate of data streaming keeps pace with computation. The second strategy relies on first copying data to the cloud destination site and then using it for computation. We evaluate these strategies in the context of a CERN LHC application [13, 14]. Multiple studies explore various data management strategies using existing storage options for clouds [52, 67, 93, 103, 150]. These general studies focus on scientific applications that process large, unstructured sets of static input data (i.e., data that is available in the cloud storage when the processing starts and remains unchanged during the computation). To the best of our knowledge, however, no previous study has considered the case of *dynamic* sets of independent pieces of input data, e.g., data streamed from a large network of sensors to the cloud for processing. Our results indicate that cloud streaming can significantly interfere with the computation and reduce the overall application performance.

An extended survey over thousands of commercial jobs and millions of machine hours of computation, presented in [87], has revealed that the execution of queries (i.e., the function applied on the live stream) is event-driven. Furthermore the analysis shows that the input data accounts only for 20 % of the total I/O, the rest corresponding to the replication of data between query services or to the intermediate data passed between them. This emphasizes that the event processing services, be they distributed, exchange large amounts of data. Additionally, the analysis highlights the sensitivity of the performance of the stream processing to the management and transfer of events. This idea is also discussed in [68], in

which the authors stress the need for a high-performance transfer system for real-time data. A similar conclusion is drawn in [22], where the issues which come from the communication overhead and replication are examined in the context of state-based parallelization. Finally, in [152], the authors emphasize the importance of *data freshness*, which improves the QoS of a stream management system and implicitly the quality of the data (QoD). All these research efforts, which complement our evaluation study, support and motivate the growing need for a high-performance system for event streaming.

Our approach in a nutshell. To address these issues, we propose a set of strategies for efficient transfers of events between cloud data centers and we introduce JetStream, which implements these strategies as a high-performance, batch-based streaming middleware. JetStream is able to self-adapt to the streaming conditions by modeling and monitoring a set of context parameters. The size of the batches and the decision on when to stream the events are controlled dynamically, based on the proposed model that characterizes the streaming latency in the context of clouds. To further improve the performance of the transfer streaming rate, we apply the approach introduced in Chapter 11. JetStream enables multi-route streaming across cloud nodes, aggregating inter inter-site bandwidth. The approach was validated on the Microsoft Azure cloud using synthetic benchmarks and a real-life scenario based on the MonALISA [117] monitoring system of the CERN LHC experiment [37]. The results show performance improvements of 250 times over today's stream approaches (i.e., individual event streaming) and 25 % over static batch streaming (i.e., using fixed batch sizes), while multi-route streaming can further triple the transfer rate.

12.1 Evaluating Strategies for Cloud Stream Processing

We evaluate today's cloud stream processing landscape with a data analysis code from an Atlas experiment at the CERN Large Hadron Collider [13, 14]. The application performs data analysis by searching in a channel where the Higgs decays into t anti-t quarks. The experimental data is collected as successive time events, an event being to the aggregated readings from the Atlas sensors at a given moment. The time series nature of the event data makes the Atlas application a stream analysis, i.e., a function is iteratively applied on the incoming event stream. Nevertheless, because the size of all the collected events is in the order of tens of Petabytes, achieving efficient processing is of significant concern. We consider two scenarios for managing stream processing on cloud facilities. In each scenario, we work with a data source and then deploy a set of compute instances (or compute VMs) on a remote cloud data center. Each compute instance is running a number of application workers that process the data. The architecture of these scenarios is presented in Figure 12.1. The goal is to assert if current streaming technologies allow high-performance real-time processing across cloud data centers.

Stream&Compute. In this scenario, data is directly streamed from the data source to the compute instances where it is ingested and processed by the worker processes. The events are processed in memory as they are streamed.

Copy&Compute. In this scenario, we allocate some persistent storage (i.e., EBS-like) and an additional instance that attaches this storage, in order to make it accessible to all

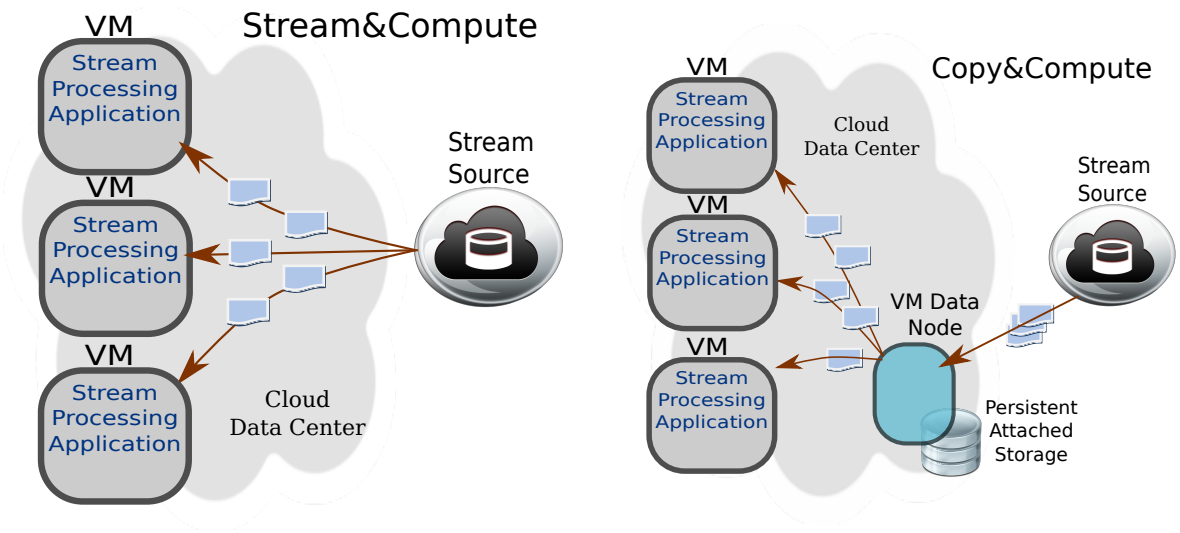


Figure 12.1: The two main cloud streaming scenarios: Stream&Compute (left) and Copy&Compute (right).

compute instances (i.e., the persistent attached storage model is detailed in Chapter 3). The streaming is split into two phases. First, we copy the data from the data source to this persistent attached storage. Second, once all data is available on the cloud site, it is locally streamed to the compute instances which begin the computation.

The advantages of Stream&Compute are the following. It provides better response time on a case by case basis where not all of event data to be computed is available up front or needs to be processed to yield a result. It has the potential to overlap computation and communication, thus potentially shortening the time to process a set of events. It uses, if needed, only the storage provided within the compute instances, thus making the computation potentially cheaper. At the same time, we note that at large-scales, network saturation can slow the data-streaming rate to the point where it no longer keeps pace with computation, potentially necessitating redistribution across different clouds. The advantage of Copy&Compute is that it relies on persistent storage, thus leading to easier repair in cases where an instance is terminated unexpectedly: while any non terminated computations will have to be rerun, the data will not have to be resent over wide-area network. Our working hypothesis is that Stream&Compute outperforms the Copy&Compute strategy. Validating this hypothesis would show that the currently-used cloud streaming engines (i.e., which stream events to be processed as they are available, in a Stream&Compute fashion) can support multi-site cloud stream processing.

The experiments were run on the Azure cloud for which we used the Azure drives [17] storage volumes to provide the persistent storage in the Copy&Compute scenario. We used Small VM Roles having 1 CPU, 1.75 GB memory, 200 GB local storage and running a CentOS 6.3 operating system. The Atlas application, we experimented with, processes small data events of similar size. Small size differences appear because the events aggregate measurements of different aspects of a phenomenon, which may or may not be detected at a given point. The number of events processed is therefore a good measure of the progress of the application. We used the following metrics for our experiments.

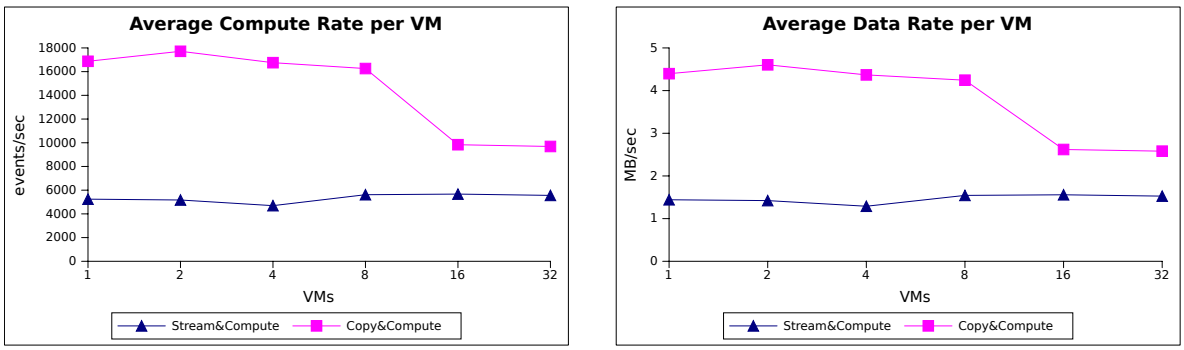


Figure 12.2: Comparing the Copy&Compute and Stream&Compute scenarios in Azure, considering average compute (left) and data (right) rates per VM.

Compute Rate which is the amount of events that are processed in a time unit.

Data Rate which is the amount of data that is acquired in a time unit. We chose for this evaluation data rate instead of event rate because of the variable sizes of the events.

Methodology. For the Stream&Compute case, we measured the compute rate by running the program with a set number of events and measuring on each compute node how long it took from start to finish of the computation. For the Copy&Compute case, the experiment involved two phases: a remote site copy followed by the computation on local data. For the first phase, we used the "time" command to measure how long a remote copy took using "scp". This was then added to the time taken by the application to complete the processing over the set of events transferred. We measured the data rate by dividing the amount of input data by the total time to complete its processing. For the Copy&Compute case, this total time included both the remote data copy and running the application. For each measurement presented in the charts, 100 independent experiments were performed. The values shown represent the averages.

We first present the average per VM of the compute rates and data rates to the application VMs. Figure 12.2 shows the results for the two scenarios for Azure. We crossed-checked the results, and observed similar trends also on the FutureGrid platform [62], using Hotel (a University of Chicago cloud configured with Nimbus version 2.10.1 [135]) and Sierra (a San Diego Supercomputing Center cloud configured with OpenStack Grizzly [140]). In addition to the data shown in the figures, we note that the results for the Copy&Compute operation show a higher variability, having a coefficient of variation (i.e., standard deviation / mean) of ∼20 %, than do the results for the Stream&Compute data, which has a coefficient of variation of ∼10 %. The remote copy phase is mainly responsible for this high variability; the variability on local dissemination is very low.

Copy&Compute vs. Stream&Compute. We see that Copy&Compute strategy offers three to four times better performance than does Stream&Compute. This is contrary to our expectation. The initial hypothesis was that Stream&Compute method is faster because of overlapping computation and communication. Therefore, considering that current stream processing systems rely on a Stream&Compute strategy, the results show that in fact they

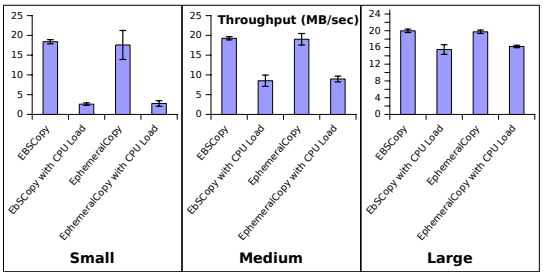


Figure 12.3: Assessing the data acquisition throughput with respect to the CPU load for three different instance types.

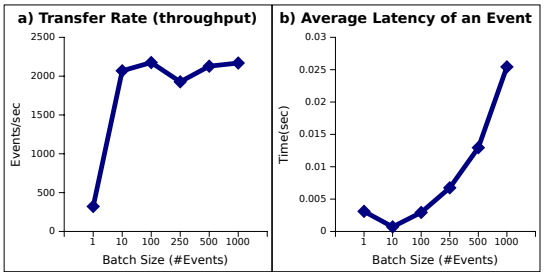


Figure 12.4: a) Transfer rate and b) average event latency for transferring events in batches between North Europe and North US Azure data centers.

cannot deliver high-performance for multi-site cloud scenarios. For example, in this experiment, the overlapping of the computation with communication is available at the level of the framework, with no particular optimizations set up on our side. Hence, similar results are expected for any framework that provides such a behavior: simply acquiring the data (i.e., event by event or small sets of them) for the next step of the computation while the available data for the current step is being processed. In addition, we see a drop in per VM average data and compute rates for the Copy&Compute case on Azure once we reach 16 VMs. This reflects the bandwidth limitation of the Azure node hosting the persistent storage, which results in contention as more nodes want to access the data. This situation could potentially be fixed by either assigning the attached storage to a larger instance or by striping the access to the data over multiple nodes.

To understand why Copy&Compute outperforms Stream&Compute, we first compared data throughput of a VM using its local ephemeral storage (as in the Stream&Compute scenario) and a VM using the persistent storage (as in the Copy&Compute scenario). All cloud storage options are detailed in Chapter 3. The comparison is done by copying large data files (0.5 GB) using the Unix "scp" command. The results, labeled EphemeralCopy and EBSCopy, respectively, are shown in Figure 12.3. We found that the throughput for small instances (the ones used in the previous experiment) in both cases is almost the same, so that the high throughput, obtained in Copy&Compute scenario, is not correlated with using the attached storage. However, when we induced a 100 % CPU load using the Unix "stress" tool with 8 threads spinning over sqrt function ("–cpu N") and spinning over the pair (memalloc/free) ("–vm N"), we saw the throughput diminish significantly — to roughly one-fifth of the initial throughput — but again roughly equally in the EBSCopy and EphemeralCopy case. In other words, the drop in throughput is correlated to the CPU usage.

Impact of latency on performance. Second, we asked the question why streaming from a remote node to application VMs (as in Stream&Compute) is more impacted by CPU activity than streaming from a local node (as in Copy&Compute). To address this question, we used the "netem" tool to increase latency between the node hosting the attached storage and application nodes of the Copy&Compute scenario to be equivalent to the remote latency used in the Steam&Compute scenario (i.e., 84 ms). The result shows that performance (i.e., both data and compute rate) drops to the level of the Stream&Compute scenario. This shows

that the difference in latency is responsible for the difference in performance. This proves that achieving high-performance for geographically distributed stream processing requires to minimize the latency while accounting for the resource usage: CPU and bandwidth.

Moving to batch-based streaming. This observation that a non real-time strategy is more efficient that a real-time one, shows that achieving real-time high-performance event streaming requires in fact new cloud-based solutions. Most of the existing stream processing engines, as the one used by CERN's Atlas applications, only focus on event processing and provide little or no support for efficient event transfers. Others even delegate the responsibility of transferring the events to the event source. Today, the typical way to transfer events is individually (i.e., *event by event*) or in small, fixed groups. This is highly inefficient, especially across WAN, due to the incurred latencies and overheads at various levels (e.g., application, technology tools, virtualization, network). A better option is to transfer events in *batches*. While this improves the transfer rate, it also introduces a new problem, related to the selection of the proper batch size (i.e., *how many events to batch?*). Figure 12.4 presents the impact of the batch size on the transfer rate, and transfer latency per event, respectively. We notice that the key challenge here is the selection of an optimal batch size and the decision on when to trigger the batch sending. This choice strongly relies on the streaming scenario, the resource usage and on the context (i.e., the cloud environment). We tackle these problems by proposing an environment-aware solution, which enables optimal-sized batch streaming of events in the clouds. To achieve this, we model the latency of the event transfer with respect to the environment, dynamically adapt the batch size to the context and enable multi-route streaming across clouds nodes.

12.2 Modeling the Streaming of Data in the Context of Clouds

Dynamically adapting the transfer of events and minimizing the latency requires an appropriate model for streaming in the cloud. The goal is to correlate the stream decision with the cloud context: variability of resources, fluctuating event generation rates, nodes and data centers routes. Performance-wise, the goal is to sustain a high transfer rate of events while delivering a small average latency per event. In this section, we introduce such a model and present the decision mechanisms for selecting the number of events to batch. This mechanisms are then leveraged as we propose a stream middleware engine, called JetStream, designed for high-performance streaming across data centers. To achieve this, JetStream supplements the list of design principles introduced in Chapter 11 with 3 stream-specific ones.

Decoupling the event transfer from processing. The event transfer module needs to be designed as a stand-alone component, decoupled from the stream processing engine (also referred to as CEP engine — see Chapter 5). We advocate this solution as it allows seamless integration with any engine running in the cloud. At the same time, it provides sustainable performance independent on the usage setup or specific architectures.

Generic solution. Building specific optimizations which target precise applications is efficient, but limits the applicability of the solution. Instead, we propose a set of tech-

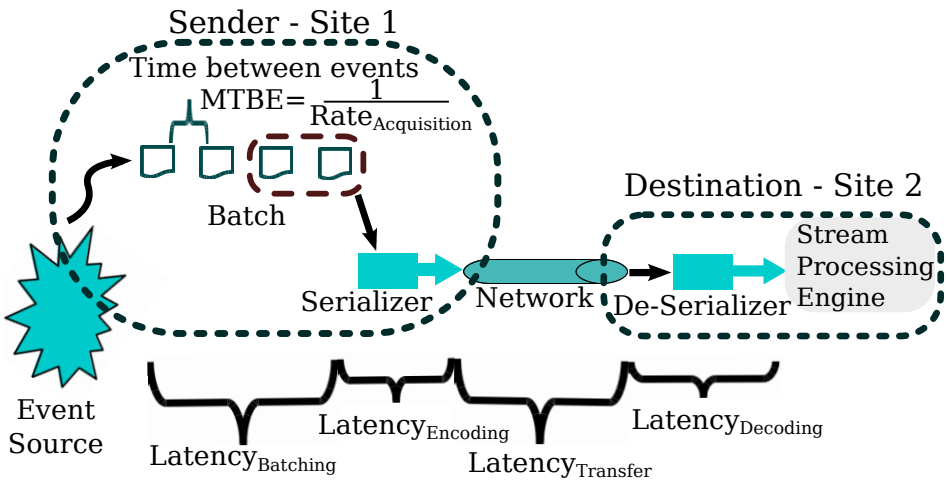


Figure 12.5: Breaking down the latency to deliver an event from source to stream processing engine across cloud nodes.

niques which can be applied in any cloud context, independent of the application semantics. JetStream does not depend on the nature of the data, nor on the query types, being thus applicable in any context.

Self-optimization. A user configuration of a system does not guarantee optimal performance, especially in dynamic environments. Moreover, when it comes to large-scale systems, the tasks of configuring and tuning services tends to become complex and tedious. The alternative is to design autonomic cloud middleware, able to self-optimize their configurations. Coupled with an economic model of resources performance, these middleware can then regulate the resource usage and enforce service-level agreements (SLAs).

12.2.1 Zoom on the Event Delivery Latency

The model we propose expresses the latency of the events based on a set of cloud parameters which can be monitored. Such a technique allows to correlate the batch size corresponding to the minimal event latency both to stream context and to environment information. We start by breaking down the latency between the source and the destination of an event in four components, depicted on Figure 12.5: creating the batch, encoding it (e.g., serializing, compression, etc.), transferring the batch and decoding it. The set of parameters able to describe the context and define the latency is: the average acquisition rate ($Rate_{Acquisition}$) or mean time between events (MTBE), the event size ($Event_{SizeMB}$), the serialization/deserialization technology, the throughput (thr) and the number of events to put in the batch (i.e., batch size - $batch_{Size}$). The goal is to determine dynamically the size of the batch, based on the latency model defined based on the other parameters.

The batching latency corresponds to the delay added when an event is waiting in the batch for other events to arrive, before they are all sent together. The parameters which

describe this latency are the average acquisition rate of the events and the number of events in the batch. As the delay depends on the position of the event in the batch (i.e., position $\times \frac{1}{\text{Rate}_{\text{Acquisition}}}$, we chose to express it as the average latency of an event. This can be computed by averaging the sum of the delays of all events in the batch:

$$\text{Latency}_{\text{batching}} = \frac{\text{batch}_{\text{Size}}}{2} \times \frac{1}{\text{Rate}_{\text{Acquisition}}}$$

Intuitively, this corresponds to the latency of the event in the middle of the sequence.

The transformation latency gathers the times to encode and to decode the batch. This applies to any serialization library/technology. The latency depends on the used format (e.g., binary, JSON, etc.), the number of bytes to convert and the number of events in the batch. To express this, we represent the transformation operation as an affine function (i.e. $f(x) = ax + b$) where a corresponds to the conversion rate (i.e., amount of bytes converted per second - time for data encoding tDe), while the b constant gives the time to write the metadata (time for header encoding tHe). The latency per event can be expressed as:

$$\text{Latency}_{\text{transformation}} = \frac{tHe + tDe \times \text{batch}_{\text{SizeMB}}}{\text{batch}_{\text{Size}}}$$

which holds both for the encoding and decoding operations. The formula can be used also to express the size of the data after the encoding operations. It only requires to replace the time-related constants with data-related ones (i.e., size of the metadata after encoding and the compression ratio). Moreover, it can be applied to other data pair transformations: compression, deduplication, encryption, etc.

The transfer latency models the time required to transfer an event between cloud nodes across data centers. To express it, we consider both the amount of data in the batch as well as the overheads introduced by the transfer protocol (e.g., HTTP, TCP) — size overhead for transport sOt and the encoding technique — size overhead for encoding sOe. Due to the potentially small size of data transferred at a given time, the throughput between geographically distant nodes cannot be expressed as a constant value. It is rather a function of the total batch size ($\text{Size}_{\text{Total}} = \text{batch}_{\text{SizeMB}} \times \text{batch}_{\text{Size}}$), since the impact of the high latency between data centers depends on the batch size. The cloud inter-site throughput - $thr(\text{Size})$ is discussed in more detail in the following section. The average latency for transferring an event can then be expressed as:

$$\text{Latency}_{\text{transfer}} = \frac{sOt + sOe + \text{batch}_{\text{SizeMB}}}{thr(\text{Size}_{\text{Total}})}$$

12.2.2 Multi-Route Streaming

Next, we focus on the streaming throughput. In order to address the issue of low inter-data center throughput, we leveraged also for streaming, the multi-route transfer strategy across intermediate nodes that we introduced in Chapter 11. For the readers' convenience, we repeat the scheme on Figure 12.6. The idea is to aggregate additional bandwidth by sending batches of events from the sender nodes to intermediate nodes within the same

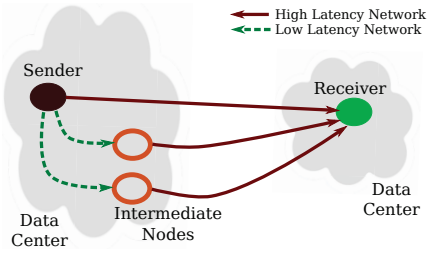


Figure 12.6: The proposed schema for multi-route streaming across cloud nodes.

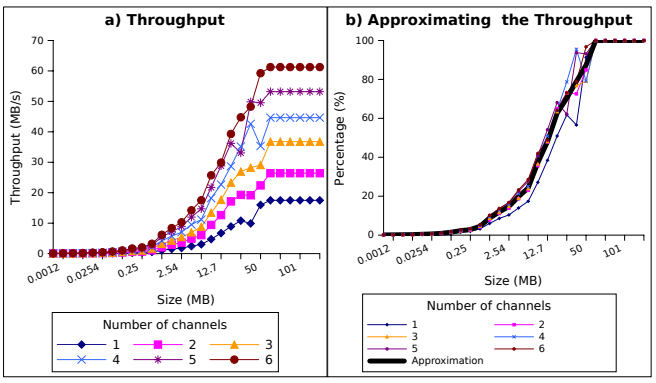


Figure 12.7: a) The throughput with respect to the number of routes across cloud Small VMs for increasing size of the data. b) Approximating the cloud throughput, independent of the number of routes, based on the averages of the normalized throughput functions for each number of routes.

deployment, which will then forward it towards the destination. To integrate this approach within our model, we extend the set of parameters that are used to characterize the stream context with the number of *channels*. This parameter gives the parallelism degree of the multi-route schema deployed for streaming, i.e., the number of nodes used on the sender site for streaming.

Multi-route throughput. Independent of the number of routes used for streaming, the value of the throughput function needs to be known to model the latency. Furthermore, considering the potentially small sizes of the data to be transferred, one needs to study the function of the throughput with respect to the size not just to measure its peak stable value. However, the performance study presented in Section 12.1 showed that increasing the usage level of the resources impacts the transfer rate. Therefore, in order to limit the number of network samples that need to be performed by the monitoring service, we approximate the throughput function. In Figure 12.7 a) we present measurements of the throughput for a number of routes between North-Central US and North EU Azure data centers. In Figure 12.7 b) we normalize these values (i.e., % of the corresponding stable throughput) and approximate them using a polynomial function. This was determined empirically that it gives a good approximation, with an error introduced due to the cloud variability of less than 15 % with respect to the approximation based on measuring the stable value. Using this approximation, the entire function can be extrapolated by measuring only the asymptotic stable throughput. This will be used as the amplitude, which multiplied with the normalized estimation, will give the throughput for any size.

Batch reordering. The downside of using multiple routes for sending batches is that the ordering guarantees offered by the communication protocol for one route does not hold anymore. This translates into batches arriving out of order due to changing conditions on the physical communication routes (e.g., packet drops, congestion, etc.). Nevertheless, it

is mandatory to maintain the integrity of the communication and avoid dropping data just because another route was faster. Hence, batches need to be reordered at the destination and the corresponding delay (i.e., *latency for reordering*) needs to be accounted for within the model. The reordering is done by buffering the batches at the destination until their turn to be delivered to the streaming engine arrives. We model the introduced latency by using the Poisson distribution ($\text{Poisson}(k, \lambda) = \frac{\lambda^k \times e^{-\lambda}}{k!}$) to estimate the probability of having k number of batches arriving before the expected batch. As we take as reference the transfer of the next expected batch, λ parameter becomes 1. This probability can then be correlated with a penalty assigned to each unordered batch. We use as penalty the latency (i.e., $\text{Latency}_{\times batch}$) incurred by having a number of batches (j) arriving out of order. This gives $\text{Poisson}(j, 1) \times j \times \text{Latency}_{\times batch}$, over which we sum in order to account for all potential number of batches arriving out of order. We denote L the maximum number of batches (e.g., 10) regarded as potentially arriving before the reference one through a channel, giving the upper limit for the summation.Finally, we sum these penalties over the number of channels, as each channel can incur its own number of unordered batches, and normalizing based on the events, as our model express everything as latency per event. The final equation that models the unordered batches arriving through all channels is:

$$\text{Latency}_{\text{reordering}} = \frac{\sum_{i=2}^{\text{channels}} \sum_{j}^{L} \text{Poisson}(j, 1) \times j \times \text{Latency}_{\times batch}}{\text{batch}_{\text{size}} \times L}$$

12.3 JetStream: Enabling High-Performance Streaming between Data Centers

The model for cloud streaming introduced in the previous section defines the average latency of transferring an event to the destination with respect to a set of parameters which can be asserted at runtime. This section details how this model is applied to select the optimal number of routes and events to batch, as well as the architecture of the JetStream middleware which implements this approach.

12.3.1 Adaptive Cloud Batching

In Algorithm 5, we present the decision mechanism for selecting the number of events to batch and the parallelism degree (i.e., channels/routes) to use. The algorithm successively estimates the average latency per event, using the formulas presented in Section 12.2, for a range of batch sizes and channels, retaining the best one. As an implementation optimization, instead of exhaustively searching in the whole space we apply a simulating annealing technique, by probing the space with large steps and performing exhaustive searches only around local optima. Depending on the magnitude of the optimal batch size, the maximal end-to-end event latency introduced by batching can be unsatisfactory for a user, as it might violate application constraints, even if the system operates at optimal transfer rates. Hence, the users can set a maximum acceptable delay for an event, which will be converted in a maximum size for the batch (Line 3). Imposing an upper bound for the batch size limits the latency to form the batch (i.e., fewer events are needed) and thus the end-to-end delay.

The selection of the number of channels is done by estimating how many batches can be formed while one is being transferred (Lines 6-8). Beyond this point, adding new channels

Algorithm 5 The selection of the optimal batch size and the number of channels to be used

1: **procedure** BATCHANDCHANNELSSELECTION
2: getMonitoredContextParameters()
3: ESTIMATE MaxBatched from [MaxTimeConstraint]
4: **while** channels $<$ MaxNodesConsidered **do**
5: **while** $\text{batch}_{\text{size}} < \text{MaxBatched}$ **do**
6: ESTIMATE $\text{latency}_{\text{batching}}$ from $[\text{Rate}_{\text{Acquisition}}, \text{batch}_{\text{size}}]$
7: ESTIMATE $\text{latency}_{\text{encoding}}$ from $[\text{overheads}, \text{batch}_{\text{sizeMB}}]$
▷ Estimate the transfer latency for 1 channel
8: ESTIMATE $\text{latency}_{\text{transfer1}}$ from $[\text{batch}_{\text{sizeMB}}, \text{thrRef}, 1\text{channel}]$
9: COMPUTE Ratio_{CPU} from $[\text{latency}_{\text{encoding}}, \text{latency}_{\text{batching}}, VM\_\text{Cores}]$
▷ Prevents idle channels
10: **if** $\text{Ratio}_{CPU} * \text{latency}_{\text{batching}} \times \text{channels} < \text{latency}_{\text{transfer1}} + \text{latency}_{\text{encoding}}$ **then**
11: ESTIMATE $\text{latency}_{\text{decoding}}$ from $[\text{overheads}, \text{batch}_{\text{sizeMB}}]$
12: ESTIMATE $\text{latency}_{\text{transfer}}$ from $[\text{batch}_{\text{sizeMB}}, \text{thrRef}, \text{channels}]$
13: ESTIMATE $\text{latency}_{\text{reordering}}$ from $[\text{channels}, \text{latency}_{\text{transfer}}]$
14: $\text{latency}_{\text{perEvent}} = \sum \text{latency}_{*}$
15: **if** $\text{latency}_{\text{perEvent}} < \text{bestLatency}$ **then**
16: UPDATE [bestLatency, Obatch, Ochannels]
17: **end if**
18: **end if**
19: **end while**
20: **end while**
21: **end procedure**

leads to idle resources and therefore decreases the cost efficiency. The condition on Line 10 prevents the system from creating such idle channels. Finally, the CPU usage needs to be accounted in the decision process, being an important factor which determines the transfer rate, as revealed by our performance study. Sending frequent small batches will increase the CPU consumption and artificially decrease the overall performance of the cloud node. This is what was observed in Section 12.1 for the Stream&Compute scenario. We therefore assign a penalty for the CPU usage, based on the ratio between the time to form a batch (a period with a low CPU footprint) and the time used by the CPU to encode it (a CPU intensive operation), according to the formula:

$$\text{Ratio}_{CPU} = \frac{\text{latency}_{\text{encoding}}}{(\text{latency}_{\text{batching}} + \text{latency}_{\text{encoding}}) \times VM\_\text{Cores}}$$

When computing the ratio of intense CPU usage, we account also for the number of cores available per VM. Having a higher number of cores prevents CPU interferences from overlapping computation and I/O and therefore does not require to prevent using small batches. With this model, the batch decision mechanism is aware of the CPU usage and on the intrusiveness of the transfer to the computation, and therefore adapts the batch size accordingly. To sum up, JetStream collects a set of context parameters (Line 2) and uses them to estimate the latency components according to the formulas presented in Section 12.2. Based on these estimations, it selects the optimal batch size and the number of channels for streaming.

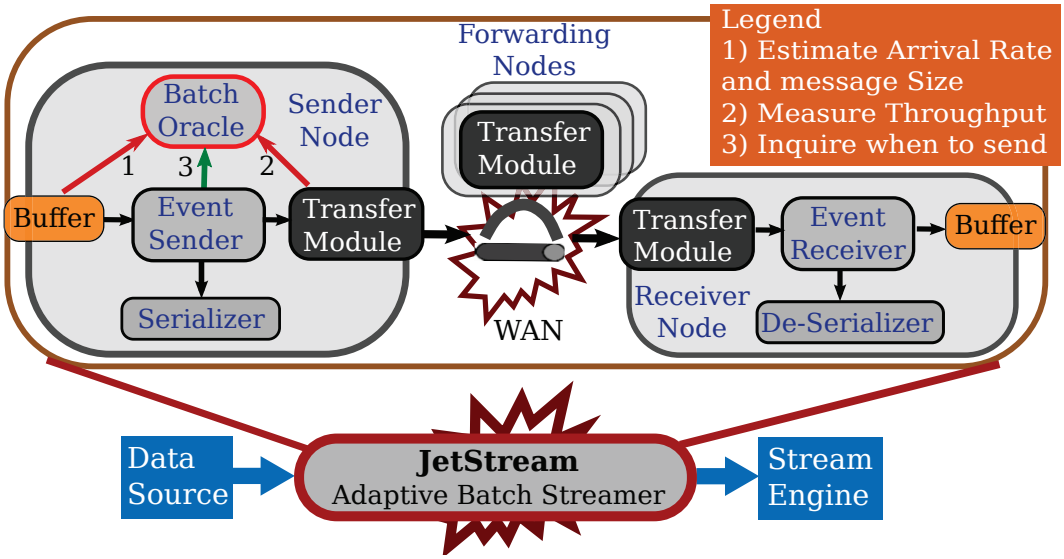


Figure 12.8: The architecture and the usage setup of the adaptive batch streamer.

12.3.2 System Architecture Overview

We designed JetStream as a high-performance cloud streaming middleware. The system is easily adoptable and ready to be used in any scenario, as it does not require changes in application semantics nor modifications or elevated privileges to the cloud hypervisor or processing engines. Its conceptual scheme is presented in Figure 12.8. The events are fed into the system at the sender side, as soon as they are produced, and they are then delivered to the stream processing engine at the destination. Hence, the adaptive-batch approach remains transparent to the system and user. The implementation of this architecture is done in C# using the .NET 4.5 framework. The distributed modules composing the system and their functionality are described below.

The Buffer is used both as an event input and output endpoint for JetStream, having thus such a component both in the sender and in receiver. The sender application or the event source simply adds the events to be transferred, as they are produced, while the receiver application (i.e., the stream processing engine) *pops* (synchronously) the events or is notified (asynchronously) when they are available. The buffer is built on 2 queues, which are used in a producer-consumer fashion. By using a separate queue for input operation to the buffer and another one for the output ones allows to reduce the number of blocked read/write operations. In this way locking is required only to swap the queues when the output (or consumer) one gets empty. The Buffer entity at the sender side is also in charge of monitoring the input stream in order to assert and report the *acquisition rate* of the events and their *sizes* in real time.

The Batch Oracle stays at the core of JetStream, as it enforces the environment-aware decision mechanism for adaptively selecting the batch size and the amount of channels to use. It implements Algorithm 5 and collects the monitoring information from the *Buffer* and the *Transfer Module*. It further controls the monitoring intrusiveness by adjusting the frequency of the monitor samples according to the observed variability.

The Transfer Module performs the multi-route streaming, using the approach presented in Chapter 11. Batches are sent in a round-robin manner through the channels assigned for the transfer. On the intermediate nodes, the role of this component is to forward the batches towards the destination. Currently, the system offers several implementations on top of TCP: synchronous and asynchronous, single- or multi-threaded. It is also in charge of probing the network and measuring the throughput and its variability as detailed in Chapter 11.

The Event Sender coordinates the event transfers by managing the interaction between modules. It queries the *Batch Oracle* about when to start the transfer of the batch. Next, it setups the batch by getting the events from the *Buffer* and adding the metadata (e.g., batch ID, streams IDs and the acknowledgment-related mechanism proposed for multi-route transfers, which is discussed in Chapter 11). The batch is then serialized by the *Serialization* module and the data transferred across data centers by the *Transfer Module*.

The Event Receiver is the counterpart of *Event Sender* module. The arriving batches are de-serialized, buffered and reordered, and delivered to the application as a stream of events, in a transparent fashion for the stream processing engine. The module issues acknowledgments to the sender or makes requests for re-sending lost or delayed batches. Alternatively, based on users' policies, it can decide to drop late batches, supporting the progress of the stream processing despite potential cloud-related failures. Users configure when such actions are performed by means of waiting times, number of batches, markers or current time increments (CTI).

Serialization/De-Serialization has the role of converting the batch to raw data, which are afterwards sent over the network. We integrate in our prototype several libraries: Binary (native), JSON (scientific) or Avro (Microsoft HDInsight), but others modules can be easily integrated. Moreover, this module can be extended to host additional functionality: data compression, deduplication, etc.

12.4 Validation and Experimental Evaluation

The goal of the experimental evaluation presented in this section is to validate the JetStream system in a real cloud setup and discuss the main aspects that impact its performance. The experiments were run in the Microsoft's Azure cloud in the North-Central US and the North EU data centers, using Small Web Role VMs (1 CPU, 1.75 GB of memory, 225 GB local storage). For multi-route streaming, up to 5 additional nodes were used within the sender deployment. Each experiment sends between 100,000 and 3.5 million events, which, depending on the size of the event to be used, translates into a total amount of data ranging from tens of MBs to 3.5 GB. Each sample is computed as the average of at least ten independent runs of the experiment performed at various moments of the day (morning, afternoon and night).

The performance metrics considered are the *transfer rate* and the *average latency of an event*. The transfer rate is computed as the ratio between a number of events and the time it takes to transfer them. More specifically, we measured, at the destination side, the time to transfer a fixed set of events. For the average latency of an event, we measured the number of events in the sender buffer, the transfer time, and reported the normalized average per event based

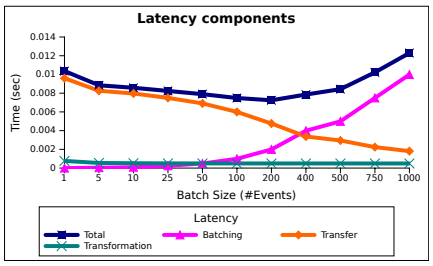


Figure 12.9: The latency components per event with respect to the batch size for a single streaming channel, for inter-site event transfers

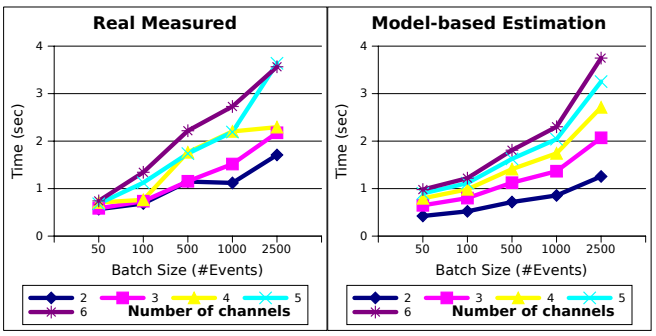


Figure 12.10: Comparing the estimation for the unordered batch latency (i.e., the penalty) with actual measurements, while increasing the number of channels used for the transfers

on the latency formulas described in Section 12.2.1. The evaluation is performed using a synthetic benchmark, that we created in order to have full control over the sizes and the generation rate of the events. The generation rates are varied between hundred of events per second to tens of thousands of events per second, as indicated by the scale of the transfer rates. Additionally, we use the monitoring readings collected by the MonALISA monitoring systems from the Alice LHC experiment to assert the gains brought by our approach in a real geographically distributed setup.

12.4.1 Accuracy of the Cloud Streaming Latency Model

We depict on Figure 12.9 the total latency per event and its components, as defined in Section 12.2, with respect to the number of batched events. Selecting the optimal size of the batch comes down to finding the value corresponding to the minimal latency (e.g., $\sim$200 for the illustrated scenario). The search for the batch size that minimizes the latency per event is at the core of the JetStream algorithm presented in Section 12.3.1. However, the selection is computed based on estimations about the environment (e.g., network throughput, CPU usage), which may not be exact. Indeed, the cloud variability can lead to deviations around the optimal value in the selection process of the amount of events to batch. However, considering that the performance of using batches with sizes around the optimal value is roughly similar (as seen in Figure 12.9), JetStream delivers more than 95 % of the optimal performance even in the case of large and unrealistic shifts from the optimum batch size (e.g., by mis-selection of a batch size of 150 or 250 instead of 200 in Figure 12.9, the performance will decrease with 3 %). The solution for further increasing the accuracy, when selecting the batch size, is to monitor the cloud more frequently, which comes at the price of higher intrusiveness and resource usage levels.

To validate the penalty model proposed for the latency of reordering batches when using multiple routes, we compare the estimations computed by our approach with actual measurements. The results are presented in Figure 12.10. The delay for unordered batches was measured as the time between the moment when an out of order batch (not the one next in sequence) arrives, and the moment when the actual expected one arrives. The experiment

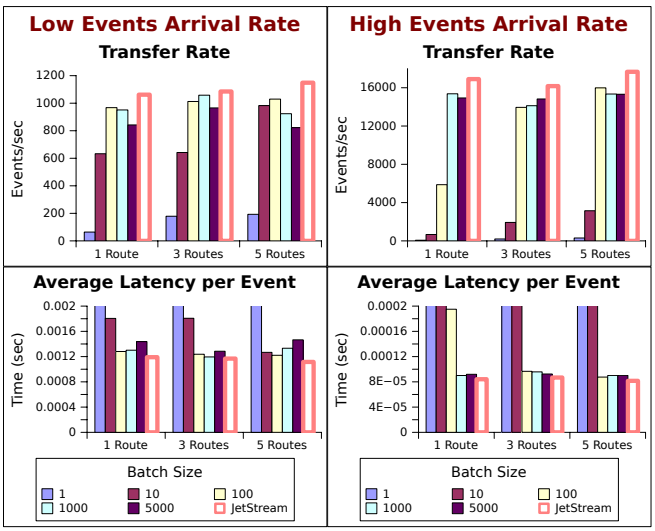


Figure 12.11: Comparing the performance (transfer rate - top and average latency per event - bottom) of individual event streaming and static batches with JetStream for different acquisition rates, while keeping the size of an event fixed (224 bytes). The performance of individual event streaming (i.e., batch of size 1) are between 50 to 250 times worse than the ones of JetStream.

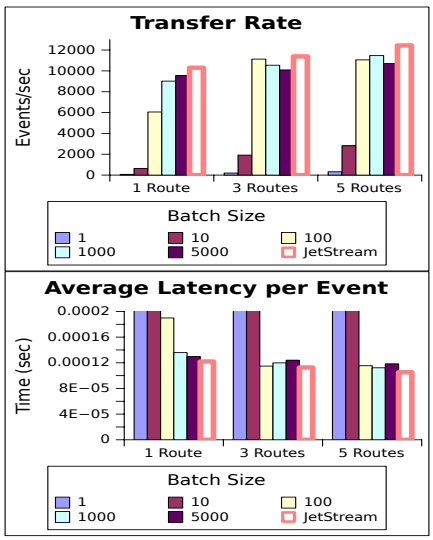


Figure 12.12: The performance (transfer rate - top and average latency per event - bottom) of individual event streaming, static batches and JetStream for events of size 800 bytes. The performance of individual event streaming (i.e., batch of size 1) are from 40 to 150 times worse than JetStream.

presents the average of this maximum unordered delay over tens of independent trials in which a fixed amount of events (i.e., 1 million events of size 2.5 KB) is transferred. We notice that the proposed model gives a good estimation of this delay, having an accuracy of 90–95 %. Errors appear because the model for the reordering delay introduced by multi-route streaming does not integrate the variability of the cloud. Yet, such errors can be tolerated as they are not determinant when selecting the number of channels to use.

12.4.2 Individual vs. Batch-Based Event Transfers

The goal of this set of experiments is to analyze the performance of individual event streaming compared to batch-based streaming between cloud data centers. For the later approach we consider both static batch sizes as well as the adaptive batch selection of JetStream. In the case of static sizes, the number of events to be batched is fixed a priori. A batch of size 1 represents event by event streaming. These setups are compared to JetStream, which implements the proposed model for adapting the batch size to the context at runtime. To this end, the evaluation is performed with different event sizes and generation rates. The experiments were repeated for different number of routes for streaming: 1, 3 and 5. We measured the transfer rates (top) and average latency per event (bottom).

The experiments presented on Figure 12.11 use an event of size 224 bytes and evaluate the transfer strategies considering low (left) and high (right) event generation rates. The first

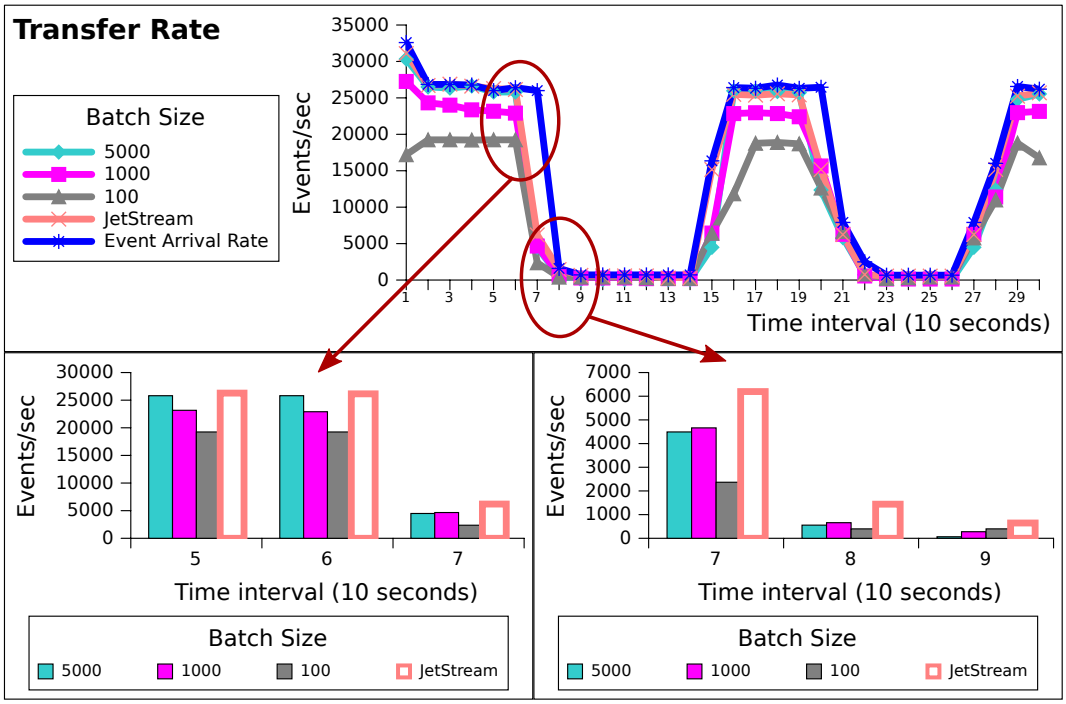


Figure 12.13: The evolution of the transfer rate in time for variable event rates with JetStream and static batches transfer strategies

observation is that batch-based transfers clearly outperform individual event transfers for all the configurations considered. The results confirm the impact of the streaming overheads on small chunk sizes and the resulting low throughput achieved for inter-site transfers. Grouping the events increases the transfer rate tens to hundreds of times (up to 250 times for JetStream) while decreasing the average latency per event. Two aspects determine the performance: the size of the batch with respect to the stream context and the performance variations of the cloud resources (e.g., nodes, network links). Static batches cannot provide the solution, as certain batch sizes are good in one context and bad in others. For example batches of size 10 deliver poor performance for 1 route and high event acquisition rate and good performance for 5 routes and low acquisition rates. Selecting the correct size at runtime brings an additional gain between 25 % and 50 % for the event transfer rate over static batch configurations (for good batch sizes not for values far off the optimal). To confirm the results, we repeated the same set of experiments for larger event sizes. Figure 12.12 illustrates the measurements obtained for event sizes of 800 bytes. The results support our conclusions, showing that JetStream is able to increase the performance with up to 2 orders of magnitude over current streaming strategies.

12.4.3 Adapting to Context Changes

The event acquisition process in streaming scenarios is not necessarily uniform. Fluctuations in the event rates of an application running in the cloud can appear, due to the nature of the data source, the virtualized infrastructure or the cloud performance variability [59]. To

analyze the behavior of JetStream in such scenarios, we performed an experiment in which the event generation rate randomly changes in time. For the sake of understanding, we present in Figure 12.13 a snapshot of the evolution of the transfer rate in which we use fine grain intervals (10 seconds) containing substantial rate changes. JetStream is able to handle these fluctuations by appropriately adapting the batch size and number of routes. In contrast, static batch transfers either are introducing huge latencies from waiting for too many events, especially when the event acquisition rate is low (e.g., batches of size 1000 or 5000 at time moment 9) or are falling behind the acquisition rate which leads to increasing amount of memory used to buffer the events not transferred yet (e.g., batch of size 100 at moment 5). Reacting fast to such changes is crucial for delivering high-performance in the context of clouds.

JetStream reduces the number of used resources by 30 % in such scenarios (e.g., the extra nodes which enable multiple route streaming), as it takes into account the stream context when scheduling the transfers. These resource savings are explained by the fact that low event acquisition rates do not require multiple route streaming as higher ones do. Additionally, as shown in [87], the fluctuations in application load have certain patterns across the week days. Hence, in long running applications, our approach will make substantial savings by scaling up/down the number of additional nodes used for transfers to these daily or hourly trends.

12.4.4 Benefits of Multi-Route Streaming

Figure 12.14 shows the gains obtained in transfer rate with respect to the number of routes used for streaming, for JetStream and for a static batch of a relatively small size (i.e., 100 events). When increasing the amount of data to be sent, multi-route streaming pays off for both strategies. This validates our decision to apply the transfer scheme proposed in Chapter 11 for bulk data transfers to streaming. By aggregating extra bandwidth from the intermediate nodes, we are able to decrease the impact of the overhead on smaller batches: batch metadata, communication and serialization headers. More precisely, a larger bandwidth allows to send more data, and implicitly, the additional data carried with each batch does not throttle the inter-site network anymore. This brings the transfer rate of smaller, and consequently more frequent, batches closer to the maximum potential event throughput. This can be observed for the static batch of size 100 on Figure 12.14, which delivers a throughput close to JetStream for a high number of routes.

With higher throughput and a lower overhead impact, the optimal batch size can be decreased. In fact this is leveraged by JetStream, which is able to decrease the end-to-end latency by selecting lower batch sizes. Hence, we conclude that sustaining high transfer rates under fixed time constraints is possible by imposing upper bounds for the batch sizes and compensating with additional streaming routes. This enables JetStream to integrate users' time constraints for maximal delay, which are integrated in the streaming decision shown in Algorithm 5 by considering a limit on the batch size.

12.4.5 Experimenting in a Real-Life Scientific Scenario

In a second phase, our goal was to assess the impact of JetStream in a real-life application. We opted for ALICE (A Large Ion Collider Experiment) [37], one of four LHC (Large Hadron

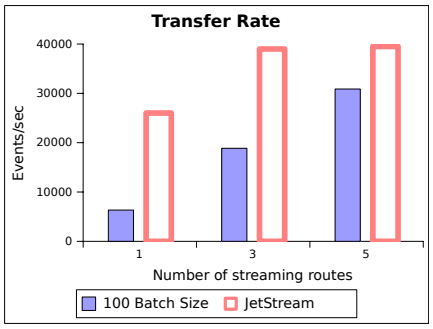


Figure 12.14: The transfer rate for an increasing number of routes used for streaming, when the batch size is fixed or adaptively chosen using JetStream.

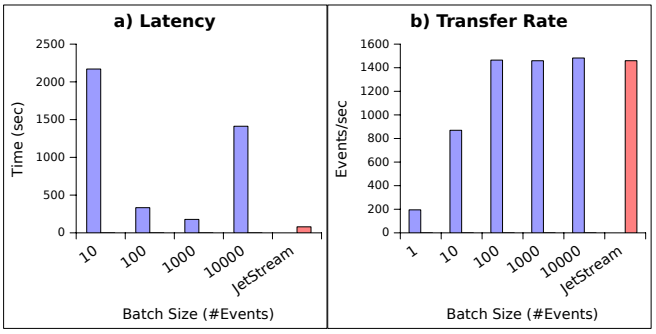


Figure 12.15: The total latency (a) and the transfer rate (b) measured when transferring 1.1 million events from MonALISA monitoring the Alice experiment. The latency for independent event transfer (batch of size 1) in a) is not represented because it would modify the scale, having a value of 30900 seconds as opposed to 80 seconds for JetStream.

Collider) experiments at CERN, as its scale, volume and geographical distribution of data require appropriate tools for efficient processing. Indeed, the ALICE collaboration, consisting of more than 1,000 members from 29 countries, 86 institutes and more than 80 computing centers worldwide, is strongly dependent on a distributed computing environment to perform its physics program. The experiment collects data at a rate of up to four Petabytes per year. Our focus, in these series of experiments, is on the monitoring information collected in real-time about all ALICE resources. We used the MonALISA [117] service to instrument and replay the huge amount of monitoring data issued from this experiment. More than 350 MonALISA services are running at sites around the world, collecting information about ALICE computing facilities, local and wide area network traffic and the state and progress of the many thousands of concurrently running jobs. This yields more than 1.1 million parameters published in MonALISA, each with an update frequency of one minute. Using ALICE-specific filters, these raw parameters are aggregated to produce about 35,000 system-overview parameters in real time. The MonALISA framework and its high-frequency updates for large volumes of monitoring data matched closely with JetStream's architecture purposes, being the reason why we chose this as a real application scenario.

JetStream vs. bad user configuration. Based on the monitoring data collected by MonALISA as of December 2013, we have replayed a sequence of 1.1 million events considering their creation times at the rate they were generated by Alice. The measurements were performed using 2 intermediate nodes located at the sender side (i.e., resulting in 3 streaming routes). Initially, the experimental setup considered 5 streaming routes. However, during the transfer of the data using JetStream, we observed that at most 3 such routes were used, as the system determined that the performance cannot be increased beyond this point. The same number of nodes is recommended if we query offline the system based on the stream context (i.e., event size and acquisition rate). The accuracy of this decision was in fact validated as our adaptive approach was obtaining the same transfer performance using 3 nodes as the static batch configurations which were using 5. Furthermore, the static configurations also

obtained the same transfer performances when switched to 3 nodes, showing that indeed this streaming context was not requiring more than 3 streaming routes. This observation shows the importance of an environment-aware adaptive approach, not subject to arbitrary human configuration choices.

Streaming monitoring data of CERN Alice experiment with JetStream. Figure 12.15 a) shows the total latency of the events at the sender and the transfer rate, Figure 12.15 b), when comparing JetStream with static configurations for various batch sizes. The transfer performance of static batches, with more than 100 events, are similar with JetStream. Considering that the generation rate of the events varies from low to high, these sub-optimal batch sizes will in fact lead to an accumulation of the events in the sender queue during the peak rates. These buffered events will artificially increase the performance, at the expense of extra memory, during the periods when the acquisition rate of events is low. All in all, this behavior will produce a stable transfer performance over a wide range of static batch sizes, as it can be observed in Figure 12.15 b). But on the other hand, it will increase the latency of the events as depicted in Figure 12.15 a). As our approach selects the appropriate batch size at each moment, it consequently reduces the amount of events waiting in the sender queue and decreases the overall latency of the events. Compared to the static batch strategies, the latency obtained with JetStream is reduced between 2.2 (100-event batches) down to 17 times (10,000-event batches).

12.5 Discussion

In this chapter, we focused on the issues of geographically distributed stream processing. The performance evaluation conducted, highlights the inefficiency of today's streaming strategies and the need for new tools able to control the resource usage. To this purpose, we proposed JetStream which leverages a novel approach for high-performance streaming across cloud data centers by adapting the batch size to the transfer context and to the cloud environment. The batching decision is taken at runtime by modeling and minimizing the average latency per event with respect to a set of parameters which characterize the context. To tackle the issue of low bandwidth between data centers, we apply the multi-route transfer approach introduced in Chapter 11 to enable multi-route streaming. JetStream was validated using synthetic benchmarks, and by using the monitoring data collected with MonALISA system based on the Alice experiment setup at CERN. The results showed that JetStream increases the transfer rates up to 250 times compared to individual event transfers. The adaptive selection of the batch size further increases performance with an additional 25 % compared to static batch size configurations. Finally, multi-route streaming triples performance and decreases the end-to-end latency while providing high transfer rates. These results show that our data management approaches can be successfully applied also in the context of streaming, to enable high-performance, real-time communication across sites.

Chapter 13

Transfer as a Service: Towards Cost Effective Multi-Site Data Management

Contents

13.1 Transfer as a Service 163
13.2 Validation and Experimental Evaluation 165
13.3 Towards a "Data Transfer Market" for Greener Data Centers 170
13.4 Discussion 174

This chapter develops the contributions published in the following paper:

- *Transfer as a Service: Towards a Cost-Effective Model for Multi-Site Cloud Data Management.*Radu Tudoran, Alexandru Costan, Gabriel Antoniu. In Proceedings of the 2014 33rd IEEE Symposium on Reliable Distributed Systems (SRDS 2014), Nara, Japan, October 2014

The global deployment of cloud data centers enables large web services to deliver fast response to users worldwide. Examples of such applications range from office collaborative tools (Microsoft Office 365, Google Drive), search engines (Bing, Google), global stock market, financial analysis tools to entertainment services (e.g., events broadcasting, games, news mining) and scientific applications [20, 43]. However, this unprecedented geographical distribution of the computation brings new challenges related to the efficient data management across sites required to maintain a global coherence between running instances for mining queries, maintenance or monitoring operations. Studies show that the inter-site traffic is expected to triple in the following years [112, 116]. Therefore, the issues related to high throughput, low-latencies, cost- or energy-related trade-offs are serious concerns not only

for cloud users, as discussed in the previous chapters, but also for cloud providers. Providing services to address these challenges by delivering high-performance data management while reducing the corresponding costs and data center energy consumption is a key milestone for the business of cloud providers and for tomorrow's cloud data centers [27].

The need for a transfer service. As of today, the cloud data management offer is limited to the cloud-provided storage (e.g., Azure Blobs, Amazon S3), provided with a rigid cost scheme (i.e., a fixed price for everything). These storage services, accessed through basic REST APIs, are highly optimized for availability, enforcing strong consistency and replication [35]. Clearly, they are not well suited for end-to-end transfers (see Chapter 11) nor for data-related services (see Chapter 10), as this was not their intended goal. In case of inter-site data movements, which is done by first storing and than reading data, the throughput is drastically reduced by the high latency of the cloud storage and the low interconnecting bandwidth between sites. Seldom, the cloud-provided storage is coupled with mechanisms such as Amazon's CloudFront [6], which uses a network of edge locations around the world to copy (cache) static content close to users. The alternative to the cloud offer are the transfer systems that users deploy on their own, i.e., *user-managed solutions*. Examples of such transfer tools, detailed in Chapter 11, are Globus Online [4, 58], Frugal [147] or StorkCloud [112]. Although such tools are more efficient than transferring data via the cloud-provided storage, they act as third-party middleware, requiring users to setup, configure and maintain complex systems, with the overhead of dedicating some of the resources to data management. Moreover, the setups of these systems tend to be done repetitively for each application scenario, which leads to a reduced reliability and re-usability. Our goal is to understand to what extent and under which incentives inter-site transfers can be externalized from users and be provided as a dedicated service by the cloud vendors.

Our approach in a nutshell. In Chapter 11, we have proposed a user-based transfer approach that was monitoring the cloud environment for insights on the underlying infrastructure. Now, we are interested to investigate how such a tool can be "democratized" and transparently offered by the cloud provider, using a *Transfer as a Service (TaaS)* paradigm. This shift of perspective arises naturally: instead of letting users optimize their transfers by making deductions about the underlying network topology and performance through intrusive monitoring, we delegate this task to the cloud provider. Indeed, the cloud owner has extensive knowledge about the network resources, which can be leveraged within the proposed system to optimize (e.g., by grouping) user transfers. Our working hypothesis is that such a service will offer slightly lower performance than a highly-optimized dedicated user-based setup (e.g., based on multi-routing through extensive use of network parallelism as we proposed in Chapter 11), but substantial higher performance than today's state-of-the-art transfer solutions (e.g., using the cloud-provided storage service or GridFTP). Our results confirm that the system is able to decrease the variability of transfers and increase the throughput up to three times compared to the baseline user options. Moreover, this approach has the advantage of freeing users from setting own systems, while providing the same availability guarantees as for any cloud managed service.

We argue that by adopting TaaS, cloud providers achieve a key milestone towards the new-generation data centers, expected to provide mixed service models for accommodating the business needs to exchange data [27]. Greenberg et al. [74] emphasize that network and

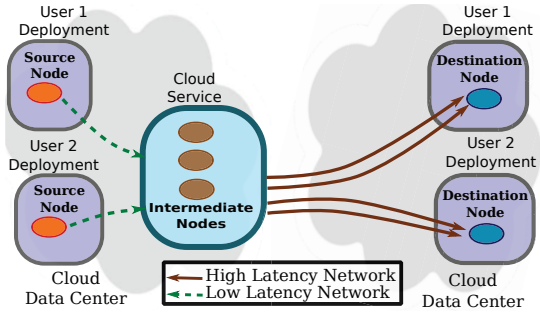


Figure 13.1: An asymmetric Transfer as a Service approach.

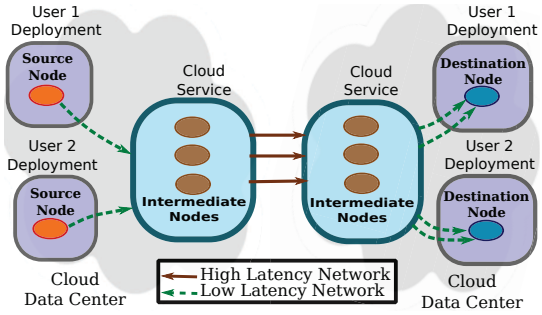


Figure 13.2: A symmetric Transfer as a Service approach.

system innovation are the key dimensions to reduce costs. Cloud providers rent the interconnecting bandwidth between data centers from Tier 1 Internet Service Providers and get discounts based on the committed transfer levels [168]. This constrains cloud vendors to the agreed data quotas as low usage or quota overpasses lead to financial losses. Therefore, coupled with the flexible pricing scheme for the service usage that we propose, TaaS can regulate the demand and facilitate the process of transferring data to a larger number of users. By enabling fast transfers through simple interfaces, as advocated by TaaS, cloud providers can therefore grow their outbound traffic and increase the associated revenues via a *data transfer market*. Finally, our results show that such a service can decrease energy consumption within a data center down to half compared to user-based transfers. Hence, our proposal for a cloud-provided transfer service enriches the cloud market offer, while increasing the profitability and efficiency of the data centers.

13.1 Transfer as a Service

The main issue with any user-managed system, let alone the ones performing data management in the cloud, is that they are not available out-of-the-box. For instance, prohibiting factors for deploying the multi-route strategy approach introduced in Chapter 11 range from the lack of user networking and cloud expertise to budget constraints. Additionally, applications might not tolerate even low intrusiveness levels for handling data in the intermediate nodes. In turn, this would limit the deployment of this mechanism only to dedicated nodes (i.e., the DataSteward approach discussed in Chapter 10). Moreover, nodes have to be provisioned in advance as scaling up VMs for short time periods to handle the transfer is currently strongly penalized by the VM startup times. From the cloud provider perspective, having multiple users that deploy multi-route transfer systems can lead to an uncontrolled boost of expensive Layer 1 ports towards the ISP [74]. Bandwidth saturation or congestion at the outer data center switches are likely to appear. The bandwidth capacity towards the Tier 1 ISP backbones, with a ratio of 1:40 or 1:100 compared to the bandwidth between nodes and Tier 2 switches, can rapidly be overwhelmed by the number of users VMs staging-out data. Moreover, activating many rack switches for such communications increases the energy consumption as demonstrated in Section 13.3.3. Our goal is to find the right trade-off between the (typically contradicting) cloud providers economic constraints and users needs.

We argue that a cloud-managed transfer service could substitute the user-based mech-

anisms without significant performance degradation. At the core of such a service lies a set of dedicated nodes within each data center, used by the cloud provider to distribute the transferred data and to forward them further towards the destination. As opposed to the DataSteward approach, the dedicated nodes are owned and managed by the cloud provider, hence, they no longer consume resources from user deployments. Building on elasticity, the service can accommodate fluctuating user demands. Multiple parallel paths are then used for data transfers, leveraging the fact that the cloud routes packages through different switches, racks and network links. In this way, the cloud service can leverage the multi-route transfer approach that we introduced in Chapter 11, and therefore it can aggregate inter-site throughput for wide-area transfers. As a result, such a system decreases the costs for users and provides better network bandwidth utilization for the cloud provider due to the multi-tenancy usage.

The proposed architecture makes the service locally available to all applications within each data center, as depicted in Figure 13.1. The usage scenario consists in: 1) applications transferring data through the intra-site low-latency links to the service (empirically determined to be at least 10 times faster than the inter-site network); and 2) the service forwarding the data across multiple routes towards the destination. The transfer process becomes transparent to users, as the configuration, operation and management are all handed to the cloud provider (*cloudified*), making it resilient to administrative errors. The service is accessed through a simple API, that currently implements *send* and *receive* functions. Despite the minimalistic API, the service solves the data management gap existing in today's *put/get* cloud-provided storage service which supports transfers only as a "side effect" of storing. Users only need to provide a pointer to their data and the destination node in order to launch a high-performance, resilient data movement. The API can be further enhanced to allow experienced users to configure several transfer parameters such as chunk size or number of routes.

Asymmetric transfer service. When the TaaS approach is available at only one endpoint of the transfer, it can be viewed as an *asymmetric service* as depicted in Figure 13.1. This is often the case within federated clouds (i.e., hybrid environments which span across multiple cloud infrastructures), where some providers may not propose TaaS. Users can still benefit from the service when migrating their data to computation instances located in different infrastructures. Such an option is particularly interesting for scientific applications which rely on hybrid clouds (e.g., scaling up the local infrastructure to public clouds). An illustrative example is the computation that led to the discovery of the Higgs boson, which was performed across the CERN and Google cloud infrastructures [43]. The main advantage with this architecture is the minimal number of hops added between the source deployment and the destination, which translates into smaller overheads and lower latencies. However, situations can arise when the network bandwidth between data centers might still not be used at its maximum capacity. For instance, applications which exchange data in real-time can have temporary lower rates of transferred packages. Taking also into account that the connection to the user destination is direct, multiplexing data from several users is not possible. In fact, as only one end of the transmission over the expensive inter-site link is controlled by the cloud vendor, communication optimizations are not feasible. To enable them, the cloud provider should manage both ends of the inter-site connection.

Symmetric transfer service. We therefore advocate the use of the *symmetric solution*, in which TaaS is available at both transfer ends. This approach makes better use of the inter-site bandwidth, and is particularly suited for transfers between data centers of the same cloud provider. With this architecture, the TaaS should be deployed on each data center. When an inter-site transfer is performed, the service local to the sender forwards the data to the destination service, which further delivers it to the destination node, as depicted in Figure 13.2. The data transfer remains transparent to the users applications, which obtain the same functionality and handle the transfers with the same API as for the asymmetric solution. This approach enables many communication optimizations which only require some simple pairwise encode/decode operations: multiplexing data from different users, compression, deduplication, etc. Such optimizations, which were not possible with the asymmetric solution, can decrease the outbound traffic, to the benefit of both users and cloud providers. Moreover, the topology of the data center, known by the cloud provider, can be leveraged with our node selection approach, presented in Chapter 10, in order to partition the nodes of the service such that the load is balanced across the Tier 2 switches. Despite the potential lower performance compared to the asymmetric solution, due to the additional dissemination step at destination, this approach has the potential of bringing several operational benefits to the cloud provider, as discussed in the following sections.

13.2 Validation and Experimental Evaluation

In this section we analyze the performance of our proposal, focusing on realistic scenarios, and compare it to user-based transfer options, presented in Chapter 11. The purpose of this evaluation is to assert the performance trade-offs arising when the service would be offered by the cloud vendor to multiple users. For ensuring comparison fairness in the number of data centers and routes used, the intermediate nodes which support the user parallel streams are deployed only at the sender side. The working hypothesis is that user-based transfers are slightly more efficient but a cloud service can deliver comparable performance with less administrative overhead, lower costs and more reliability guarantees. The experiments were performed using Small and xLarge VMs on the Microsoft Azure cloud, using two data centers: North-Central US and North EU, with data being transferred from US towards EU. Considering the time zone differences between the sites, the experiments are relevant both for typical user transfers and for cloud maintenance operations (e.g., bulk backups, inter-site replication). The latter operations enable the cloud providers to tune the TaaS approach according to the hourly loads of data centers, as discussed in [116].

The measurements are performed by repeatedly transferring data chunks of 64 MB each from the memory of the VMs. The intermediate nodes, which support the parallel streams, handle the data entirely in memory, both for user and cloud transfer configurations. For all experiments in which the number of resources is scaled, the amount of transferred data is increased proportionally, such that a constant amount of data is handled per intermediate node. The throughput is computed at the receiver side by measuring the time to transfer a fixed amount of data. Each sample is the average of at least 100 independent measurements. Finally, this evaluation is performed in the cloud user-space, which is not the one a provider would actually use. In a real deployment, the machines used by the vendors to host the service would most likely be powerful physical nodes with specific network prop-

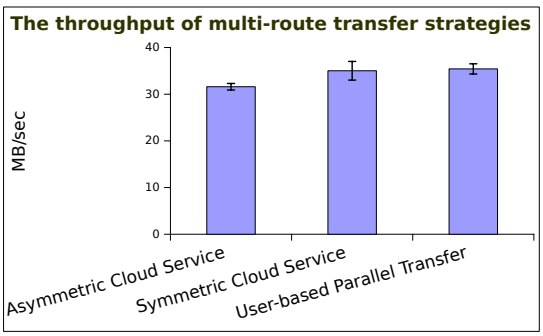


Figure 13.3: The average throughput and the standard deviation with different transfer options with 5 intermediate nodes used to multi-route the packets.

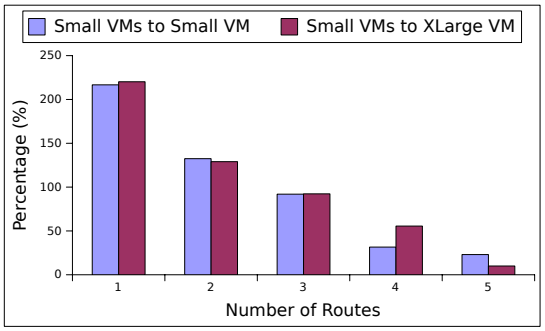


Figure 13.4: Relative variation of throughput performance of inter-site transfers with respect to the number of routes.

erties and higher performance than the xLarge, 1 GB Ethernet nodes used here. This is an advantage, since the results of our proposal in a real environment are likely to be better than the presented ones.

13.2.1 Evaluating the Inter-Site Transfer Options

Focus on transfer performance. We present in Figure 13.3 the comparison between the average throughput of the cloud transfer service and the user-based, multi-route option. The experimental setup consists of 5 nodes for each transfer service dedicated for data handling. The *asymmetric solution* delivers slightly lower performance (∼16 %) than a user-based, multi-route system. The first factor causing this performance degradation is the overhead introduced by the load balancer that distributes all incoming requests of a service between its nodes (here the requests from the application to the TaaS nodes). The second factor is the placement of the VMs in the data center. For user-based deployments, the sender node and the intermediate nodes are closer rack-wise, some of them being even in the same fault domain (i.e., belonging to the same rack switches). This translates into less congestion in the switches during the first phase of the transfer when data is locally replicated to the intermediate nodes. For the cloud-managed transfers, the user node that initializes the transfer and the cloud dedicated transfer nodes belong to distinct deployments, meaning that they are farther apart with no proximity guarantees. The *symmetric solution* is able to compensate for this performance degradation with the extra nodes at the destination site. After all, this option uses twice as many nodes as the asymmetric cloud service or user-based, multi-route strategy. The overhead of the additional hop introduced with this symmetric architecture is neutralized when additional resources are provisioned by the cloud provider. The observation opens the possibility for differentiated cloud-managed transfer services in which different QoS guarantees are proposed and charged differently.

Focus on transfer stability. Next, we focus on the performance variability expected with such multi-route transfers, which are displayed on Figure 13.4 through the coefficient of variation (i.e., standard deviation/average%). Contrary to our initial expectation, using multiple paths decreases the otherwise large performance variability of inter-site transfers. This re-

sult is explained by the fact that with multiple routes, performance drops on some links are compensated by bursts on others. The overall cumulative throughput, perceived by an application in this case, tends to be more stable. This observation is particularly important for scientific applications which build on predictability and stability of performance. Furthermore, it shows that this TaaS approach can meet the reliable performance SLA that cloud providers seek for their services.

13.2.2 Dealing with Concurrency

Impact of multi-tenancy on transfer performance. The experiment presented in Figure 13.5 depicts the throughput of an increasing number of applications using the transfer service in a configuration with 5 intermediate nodes per service. The goal of this experiment is to assess whether a sustainable QoS can be provided to user's applications which concurrently access the TaaS system. Not surprisingly, an increase in the number of parallel applications from 1 to 5 decreases the average transfer performance per application with 25 %. This is caused by the congestion in the transfers to the cloud service nodes and by the limit in the inter-site bandwidth that can be aggregated by these nodes. While this might seem a bottleneck for providing TaaS at large-scale, it is worth zooming on the insights of the experiment to learn how such a performance degradation can be alleviated. We have scaled the number of clients up to the point where their number matches the number of nodes used for the transfer service. Hypothetically, we can consider having 1 node from the transfer service per client application. At this point the transfer performance delivered by the service per application is reduced, but asymptotically bounded to the throughput loss of 25 % compared to the situation where only one application was accessing the service and all its 5 nodes where serving it. This shows that by maintaining a number of VMs proportional to the number of applications accessing the service, TaaS can be a viable solution and that it can in fact provide high-performance for many applications in parallel. Moreover, in real-life usage, we can expect potential variability of applications request rates to average out and thus to decrease the global pressure on the service.

We further notice that with increased concurrency, the performance of the symmetric solution drops more than in the case of the asymmetric one. This demonstrates that the congestion in handling data packets in the service nodes is the main cause of the performance degradation, since its effects are doubled in the case of the symmetric approach. Finally, the aggregated throughput achieved by the applications using the transfer service is equivalent with each of them using 3 dedicated nodes to handle the transfer. This shows that the transfer performances achieved with 15 user nodes can be matched by 5 or 10 nodes with the asymmetric or the symmetric TaaS solutions. Hence, deploying such services would make the inter-site transfers more energy-efficient and the data centers greener as it enables to reduce the number of resources required to perform data management operations. We further develop this analysis in Section 13.3.3.

Impact of CPU load on transfer performance. The experiments considered so far that applications dedicate the intermediate nodes to manage data as for the DataSteward approach (Chapter 10). However, not all applications afford to fully dedicate several nodes just for performing transfers, in which case they are collocating data-related operations with the computation as proposed by TomusBlobs in Chapter 7. It is interesting to analyze to what

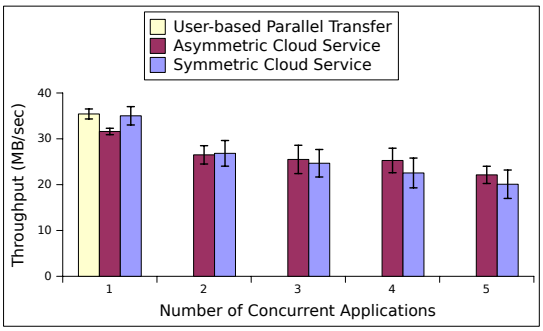


Figure 13.5: The average throughput and the corresponding standard deviation for an increasing number of applications using the cloud transfer service concurrently.

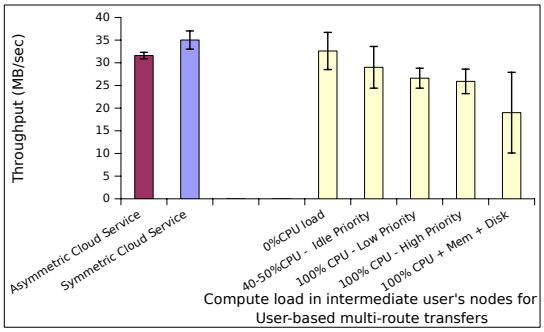


Figure 13.6: Comparing the throughput of the cloud service against user-based, multi-route transfers, using 4 extra nodes. The measurements depict the performance while the intermediate nodes are handling different CPU loads.

extent the computation load from the intermediate nodes can impact the performance of user-based transfers. We present in Figure 13.6 the evolution of the throughput when the computation done in the intermediate nodes has different CPU loads and execution priorities. All 100 % CPU loads were induced using the standardized "HeavyLoad" tool [88], while the 40 %-50 % load was generated using system background threads performing local operations.

Two main observations can be made based on the results displayed on Figure 13.6. First, the throughput is reduced between 20 % to 50 % when the intermediate nodes are performing other computation in parallel with the transfers. This illustrates that the I/O inter-site throughput is highly sensitive to CPU usage level. This observation is consistent with our previous findings related to the I/O behavior for streaming strategies, discussed in Chapter 12. Additionally, it complements the I/O analysis discussed in [50] for storing data in the context of HPC or in [63] for the TCP throughput with shared CPUs between several VMs. Second, the performance obtained by users under CPU load is similar, or even worse, to the one delivered by the transfer service under increased concurrency (see the previously discussed results from Figure 13.5). This gives a strong argument for many applications running in the cloud to migrate towards a TaaS offered by the cloud provider. Doing so, these applications are able to perform high-performance transfers while discharging their VMs from auxiliary tasks other than the computation for which they were rented for.

13.2.3 Inter-Site Transfers for Big Data

In the next experiment, larger sets of data ranging from 30 GB to 120 GB are transferred between sites, using the cloud- and the user-managed approaches. The goal of this experiment is to understand the viability of the cloud services in the context of geographically distributed Big Data applications. The results are displayed on Figures 13.7 and 13.8. The key differences from the previous experiments are that longer transfer periods will experience more changes (i.e., both drops and peaks in the cloud performance) and that resources are stressed harder (e.g., the buffers from switch and intermediate node, the memory and

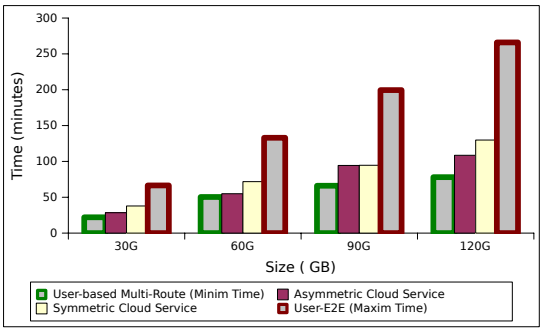


Figure 13.7: The time to transfer large data sets using the available options. The user default Endpoint-to-Endpoint (E2E) option gives the upper bound, while the User-Based, Multi-Route offers the fastest transfer time. The cloud services, each based on 5 nodes deployments, provide intermediate performances, closer to the lower bounds.

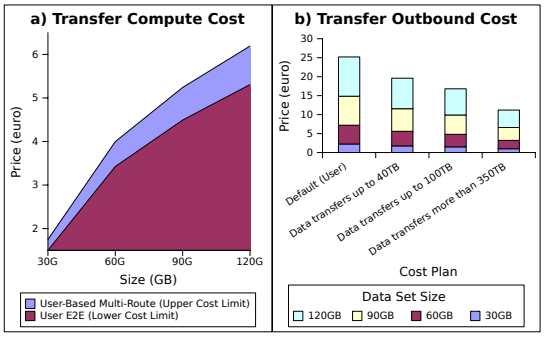


Figure 13.8: The cost components corresponding to the transfer of 4 large data sets. a) The cost of the compute resources which perform the transfers, with their lower and upper bounds. b) The cost for the outbound traffic with respect to the available cost plans offered by the cloud provider.

network cards of VMs). This experiment is relevant both to users and cloud providers since it offers concrete cost incentives (e.g., money, time) for performing large data movements in the cloud. To the best of our knowledge, there are no previous performance studies about data management capabilities of cloud infrastructures across data centers.

Focus on transfer times. Figure 13.7 presents the transfer times in 4 scenarios. The baseline option, which is given by simple user endpoint-to-endpoint transfer, provides very poor performance due to the low bandwidth between the data centers. In fact, the resulting times can be considered as upper bounds of user-based transfers, as we do not consider the usage of the cloud storage, which gives even slower performances as shown in Chapters 10 and 11. The user-based, multi-route approach is the fastest, and its performance can be considered as the lower bound for the transfer times. In-between, the cloud transfer services are up to 20 % slower than a user-based, multi-route approach but more than 2 times faster than the user baseline option, which, as discussed in previous chapters, is the typical option used today.

Focus on transfer costs. In Figure 13.8, we depict the corresponding costs of these transfer scenarios. The costs is divided in two components: the *compute cost*, paid for leasing the corresponding number of VMs for the transfer period and the *outbound cost*, which is charged based on the amount of data exiting the data center. Despite taking the longest time, the compute cost of the user-based endpoint-to-endpoint is the smallest among all options, as it only uses 2 VMs (i.e., sender and destination). On the other hand, user-based multi-route transfers are faster but at higher costs, which come with the usage of the extra VMs, as explained in Section 11.3.3. The outbound cost only depends on the data volume and the cost plan. As the inter-site infrastructure is not the property of the cloud provider, part of this cost represent the ISP fees, while the difference is accounted by the cloud provider. The real cost (i.e., the one charged by the ISP) is not publicly known and depends on business agreements

between the companies. However, we can assume that this is lower than the minimum price charged to the cloud customers, giving thus a range in which the outbound price can potentially be adjusted. Combining the observations about the current pricing margins for transferring data with the performance of the cloud transfer service, we argue that cloud providers should propose TaaS as an efficient transfer mechanisms with *flexible prices*. In this way, cloud vendors can use TaaS to regulate the outbound traffic of data centers, reducing their operating costs, and minimizing the idle bandwidth. In the next section we introduce such a pricing scheme, which enables a *data transfer market* for clouds.

13.3 Towards a "Data Transfer Market" for Greener Data Centers

In this section we discuss the advantages brought by the proposed cloud service for inter-site data transfers. From the users perspective, TaaS can offer a transparent and easy-to-use method to handle large amounts of data. We demonstrated that the service can sustain a high throughput, close to the one achieved by users when renting at least 4–5 extra VMs and dedicate them for handling data. Besides avoiding the burden of configuring and managing extra nodes or complex transfer tools, the performance-cost ratio can be significantly increased. From the cloud providers points of view, such a service would give an incentive to increase customer demand and bring competitive economic and energy advantages. We further elaborate in this direction by proposing a flexible pricing scheme, discussed in Section 13.3.1. From the economical point of view, this enables cloud vendors to regulate the demand in order to provide sustainable QoS and minimize the idle resource periods. Finally, Section 13.3.3 presents the analysis of our approach from the energy consumption point of view. We show that from this point of view, the cloud transfer service can significantly decrease the energy, making the data center more green.

13.3.1 A Flexible Price Scheme for a Transfer Market

In our quest for a viable pricing scheme, we start by defining the cost structure of the transfer options: TaaS and user-based. The price is composed from the outbound traffic cost and the computational costs. The *outbound cost* structure is identical for all transfer strategies while the *computational cost* is specific to each option.

Outbound Cost:

$$Size \times Cost_{outbound}$$

where $Size$ is the volume of transferred data and the $Cost_{outbound}$ is the price charged by the cloud provider for the traffic exiting the data center.

Computational Cost:

User-managed Endpoint-to-Endpoint option:

$$time_{E2E} \times 2 \times Cost_{VM}$$

where $time_{E2E}$ is the time to transfer data between the sender and the destination VMs, which give the 2 VMs accounted in the formula. To compute the total cost, this is multiplied by the renting price of a VM: $Cost_{VM}$.

User-managed Multi-Route option:

$$time_{UMR} \times (2 + N_{extraVMs}) \times Cost_{VM}$$

where $time_{UMR}$ is the time to transfer data from the sender to the destination using $N_{extraVMs}$ extra VMs, resulting in 2+$N_{extraVMs}$ VMs in total . As before, the cost is obtained by multiplying with the VM cost.

TaaS option:

$$time_{CTS} \times 2 \times Cost_{VM} + time_{CTS} \times service_{compute_{cost}}$$

where $time_{CTS}$ is the transfer time and $service_{compute_{cost}}$ is the price to be charged by the cloud provider for using the transfer service. Hence, the TaaS cost is composed from the price for leasing the sender and destination VMs, giving as before the 2 VMs accounted in the formula, plus the price for using the service for the period of the transfer.

The computation cost paid by users ranges from the cheapest Endpoint-to-Endpoint option to the high-performance, but more expensive, user-managed, multi-route transfers. Considering also the observations made in Section 13.2.3, these costs can be used as lower and upper margins for defining a flexible pricing schema, to be charged for the time the cloud transfer service is used (i.e., $service_{compute_{cost}}$). Moreover, by defining the cloud service cost within these limits, this is correlated with its actual delivered performance, which is between the user-based options. To represent the $service_{compute_{cost}}$ as a function within these bounds, we introduce the following gain parameters, that describe the performance proportionality between transfer options: $time_{E2E} = a \times time_{UMR} = b \times time_{CTS}$ and $time_{CTS} = c \times time_{UMR}$. Based on the empirical observations shown in Section 13.2, we can instantiate the parameters with the following values: $a = 3$, $b = 2.5$ and $c = 1.2$. By rewriting the initial cost equations and simplifying terms, we obtain in Equation 13.1 the cost margins for the $service_{compute_{cost}}$.

$$2 \times Cost_{VM} \times (b-1) \leq service_{compute_{cost}} \leq Cost_{VM} \times \frac{2 + N + 2 \times c}{c} \tag{13.1}$$

With Equation 13.1, we demonstrate that a flexible cost schema is indeed possible. Varying the cost within these margins, a *data transfer market* for inter-site data movements can be created, providing the cloud provider with the mechanisms to regulate the outbound traffic and the transfers demand, as discussed next.

13.3.2 The Data Transfer Market

Offering diversified services to customers in order to increase usage and revenues are among the primary goals of the cloud providers. We argue that these objectives can be fulfilled by creating a *data transfer market*. This can be implemented based on the proposed cloud transfer service offered at SaaS level with reliability, availability, scalability, on-demand provisioning and pay-as-you-go pricing guarantees. In Equation 13.1, we defined the margins within which the service cost can be varied. We illustrate on Figure 13.9 these flexible prices for the two TaaS declinations (symmetric and asymmetric). The values are computed based on the measurements for transferring the large data sets presented in Section 13.2.3. The cost is normalized and expressed as the price to be charged for the service usage (i.e., the

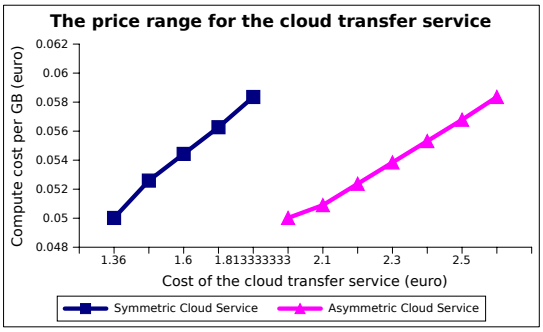


Figure 13.9: The range in which the cloud services price can be varied.

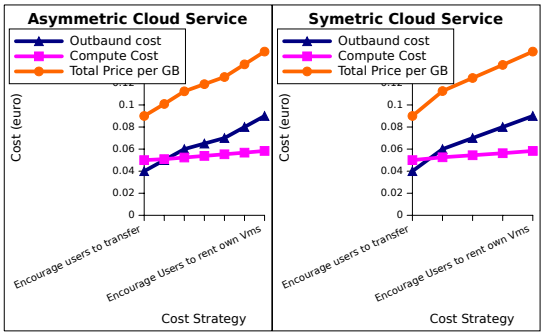


Figure 13.10: Aggregating the cost components, outbound traffic cost and computation cost, into a unified cost schema for inter-site traffic.

compute cost component) to transfer 1 GB of data. A conversion between the *per hour* VM pricing model and *per GB* usage of the TaaS service is possible due to the stable performance delivered by this approach. The main advantage of having the price expressed in terms of data size is that users can easily reason in this way. Additionally, they can make better estimates about the operation costs of their applications, as both business and scientific users typically have a good idea about the data volumes processed.

Cost boundaries. The minimal and maximal values in Figure 13.9 correspond to the user-managed solutions (i.e., endpoint-to-endpoint and multi-route). Between these margins, the cloud vendor can model the TaaS price based on a range of discrete values, as illustrated in in Figure 13.9. The two TaaS declinations have different pricing schemes due to their performance difference, with the symmetric one having slightly lower performance and consequently a lower price. As for the outbound cost, the assumption we made is that any outbound cost scheme offered today brings profit to the cloud provider. Hence, we propose to extend the flexible usage pricing to integrate also this cost component, as illustrated in Figure 13.10. The main advantage is that the combined cost gives a wider margin in which the TaaS price can be adjusted. Additionally, it allows cloud providers to propose a unique cost scheme instead of charging users separately for the service usage and for the outbound traffic. This is an important aspect as it facilitates the user task to understand and adopt the service.

Regulating inter-site data traffic. A critical benefit of setting up a *data transfer market* for the TaaS is that it enables cloud providers to regulate the data centers traffic. Reducing idle bandwidth periods can simply be achieved by decreasing the price towards the lower bound which in turn will encourage users to send data. Price drops would attract users which otherwise would need to rent and dedicate 4–5 VMs to perform transfers with equivalent performance. Building on this cost approach and complementing the work described in [116], applications could buffer in the VM storage the less urgent data and send it in bulks only during the discounted periods. In this way, the price can automatically regulate the priority of transfers, preventing low priority data transfers to throttle urgent ones. On the other hand, when many users send data simultaneously, independently or using TaaS, the

overall performance decreases due to switch contentions and network bottlenecks. Moreover, the peak usage of outbound traffic from the cloud towards the ISPs grows, which leads to lower profit margins and penalty fees for potentially exceeding the SLA quotas [83, 151, 168]. It is in the interest of the cloud providers to avoid such situations. With the proposed flexible pricing scheme, they have the means to react to such situations by simply increasing the service usage price. With the price approaching the ones of user multi-route option, the demand can be temporarily decreased. At this point, it becomes more beneficial for users to get their own VMs to handle data, as within the same budget, they can achieve higher transfer performance through the user-based multi-route approach that we introduced in Chapter 11.

Adjusting the price strategy on the fly, following the demand, produces a win-win situation for users and cloud providers. Clients have multiple services with different price options, allowing them to pay the desired cost that matches their targeted performance. Cloud providers increase their revenues by outsourcing the inter-site transfers from clients and by controlling the traffic. Finally, TaaS can act as a proxy between ISPs and users, protecting the latter from price fluctuations introduced by the former; after all, cloud providers are less sensitive to price changes than users are, as discussed in [168].

13.3.3 The Energy Efficiency of Data Transfers

The energy consumption of data centers is among the critical challenges of large-scale infrastructures such as the clouds. When breaking the operating costs of a cloud data center, the authors of [74] find that "*over half the power used by network equipment is consumed by the top of rack switches*". Such a rack switch connects around 24 nodes and has an hourly energy consumption of about 60 W/h, while a server node consumes about 200 W/h [10]. Our goal is to assess and compare the energy consumed in a data center where data transfers are performed using user-based multi-route setups (E_{UMR} in Equation 13.2) or using a cloud-provided TaaS (E_{CTS} in Equation 13.3). The comparison considers N_{App} applications, each using $N_{extraVMs}$ extra nodes when deploying the user-based multi-route transfer approach. For simplicity, we use the average transfer time (*time*) of applications.

$$E_{UMR} = (\frac{N_{App} \times (2 + N_{extraVMs})}{24} \times 60\frac{W}{h} + N_{App} \times (2 + N_{extraVMs}) \times 200\frac{W}{h})) \times time \quad (13.2)$$

$$E_{CTS} = (\frac{N_{App} \times 2}{24} \times 60W/h + N_{App} \times (2 \times 200W/h) + \frac{Nodes_{TaaS} \times 60W/h}{24} + Nodes_{TaaS} \times 200W/h) \times time \times c \quad (13.3)$$

Equation 13.2 defines the total energy consumed in the user-based scenario by the application nodes and switches. The first part of the equation corresponds to the energy used by the rack switches in which the applications nodes are deployed, while the second part gives the power used by the nodes. In Equation 13.3, we present the energy consumed when the applications use the TaaS to perform their transfers. The total energy in this case is the sum

of: 1) the energy used at the application side (i.e., the sender, destination nodes and the rack switches they activate); and 2) the energy consumed at the transfer service side, by the nodes which operate it (i.e., $Nodes_{TaaS}$) and the switches connecting them.

$$\frac{User - based_{energy}}{TaaS_{energy}} = \frac{N_{App} \times (2 + N_{extraVMs})}{(2 \times N_{App} + Nodes_{TaaS}) \times c} \quad (13.4)$$

Energy savings. The final step of this analysis is to compare the two scenarios. We do so by computing the amount of extra energy used when each user is handling his data on its own and not via the cloud service. We represent this by the ratio between the energy consumed in the two scenarios, i.e., $\frac{User-based_{energy}}{TaaS_{energy}}$, which is defined in Equation 13.4. The formula defined in Equation 13.4 is generic and can be applied by any cloud provider to estimate its energy gains if adopting our TaaS approach. When we apply it to the configurations used in the evaluation section ($N_{extraVMs} = 5$, $N_{App} = Nodes_{TaaS}$ and $c = 1.2$), we notice that twice more energy is consumed if the transfers are done by users. This result shows that by adopting our approach for a transfer service, the cloud providers can expect significant reduction of the energy consumed in their data centers for handling data.

13.3.4 Reliability

A cloud managed transfer service has the advantage of being always available, in line with the reliability guarantees of all cloud services. Requests for transfers are carried over network paths that the cloud provider constantly monitors and optimizes for both availability and performance. This allows to quickly satisfy peaks in demand with rapid deployments and increased elasticity. Cloud providers ensure that a TaaS system incorporates service continuity and disaster recovery assurances. This is achieved by leveraging a highly available load-balanced dedicated nodes-farm to minimize downtime and prevent data losses, even in the event of a major unplanned service failure or disaster. Predictable performance can be achieved through strict uptime and SLAs guarantees.

User managed solutions typically involve hard-to-maintain scripts and unreliable manual tasks, that often lead to discontinuity of service and errors (e.g., incompatibility between new versions of some building blocks of the transfer framework). These errors are likely to cause VM failures and, currently, the period while a VM is stopped or is being rebooted is charged to users. With a TaaS approach, both the underlying infrastructure failures and the user errors are isolated from the transfer itself: they are transparent to users and are not charged to them. This allows to automate file transfer processes and provides a predictable operating cost per user over a long period.

13.4 Discussion

In this chapter we introduced a new paradigm, *Transfer as a Service*, for handling large-scale data movements in federated cloud environments. The idea is to delegate the burden of data transfers from users to the cloud providers, who are able to optimize them through their

extensive knowledge on the underlying topologies and infrastructures. We propose a prototype that validates these principles through the use of a set of dedicated transfer VMs that further aggregate the available bandwidth, leveraging our approach for multi-route transfers across geographically distributed cloud sites. We show that TaaS is able to effectively leverage this transfer approach in a multi-tenant context, sustaining high-performance transfers up to 3 times faster than current state-of-the-art user tools. At the same time, it enables a reduction to half of the energy fingerprint for the cloud providers, while it sets the grounds for a *data transfer market*. As a result, cloud vendors can regulate the data movements, decreasing the periods with idle traffic while ensuring sustainable QoS transfer performance. In the same time, such a cloud managed transfer service has the advantage of being always available, in line with the reliability guarantees of all cloud services. With a TaaS approach, users are relieved from the burden of configuring and managing own-deployed transfer tools. Both the underlying infrastructure failures and the user errors are isolated from the transfer itself and become transparent to users data movements. This allows to automate file transfer processes and decrease the operating costs over the long period for Big Data processing in the clouds.

Part IV

Conclusions and Perspectives

Chapter 14

Conclusions

Contents

14.1 Achievements **179**
14.2 Perspectives **182**

The challenges brought by the Big Data paradigm reveal several limitations of the current cloud data management services. Performance trade-offs inherent in current virtualization technologies, rigid cost schemes and lack of functionalities are some of the issues identified in this work. These prevent Big Data applications to fully benefit from the advantages brought by cloud computing such as elasticity and scalability.

In this thesis we addressed these issues by proposing several solutions that enrich the cloud data-management offer, and that enable high-performance processing at large-scale, within and across cloud data centers. We demonstrated the advantages of our contributions by applying them to state-of-the-art processing solutions, and consequently improving their overall performance, for tackling real scientific applications. In this chapter, we present an overview of the main achievements and discuss the perspectives that this research opens for Big Data management on clouds.

14.1 Achievements

TomusBlobs: Federating Virtual Disks as a High-Performance Alternative to the Cloud Storage

The cloud-provided storage service is the default solution today for managing data on the clouds. However, when it comes to large-scale scientific processing such as MapReduce or workflows, this storage service is unable to exploit the processing semantics or data access patterns, and it is subject to low throughput and high latencies.

As an alternative, we proposed TomusBlobs, which provides a concurrency-optimized, PaaS level cloud storage leveraging the free virtual disks of the compute nodes. For data-intensive applications consisting of a large set of VMs, it provides a uniform storage platform that supports processing optimizations and seamless scaling. Moreover, it reduces the overall costs by replacing the payable cloud storage. This approach was validated for both MapReduce and workflow processing. Regarding MapReduce applications, we built on top of TomusBlobs an efficient framework, that leverages and optimizes the MapReduce paradigm for scientific computation. Regarding workflow processing, we extended our model to integrate application semantics and made it available within the Microsoft Generic Worker engine. The evaluation showed that TomusBlobs provides substantial performance improvements for managing data at large-scale compared to cloud-provided storage service or default state-of-the-art solutions. Moreover, we showed that TomusBlobs can provide additional features and optimizations in both contexts, such as elastic scaling and scheduling the computation to the cloud nodes, based on data access patterns and I/O context.

Benefits Validated through Real-life Big Data Scenarios and Scientific Discovery

One of the main objectives of our work was to propose data management solutions that facilitate and enable scientific discovery using cloud platforms. To this purpose, we used several real-life scenarios to validate and evaluate the benefits of our contributions: the CERN LHC Atlas application and experimental data, the MonAlisa monitoring system of the CERN LHC Alice experiment or the BLAST bio-informatic analysis. Additionally, we particularly focused on the A-Brain application to validate the benefits brought for large-scale Big Data analysis. The A-Brain pioneer analysis aims to joint neuro-imaging with genetic analysis in order to explain and provide scientific evidence of significant links between brain regions and genetic data.

To accommodate its large computation and data needs, we scaled the TomusBlobs solution to 1000 cores across 3 Azure data centers, which ran for 2 weeks and consumed more than 200,000 compute hours. This served in a collaboration with a bio-informatics team, where the consortium provided the first statistical evidence of functional signals in a failed-stop task in basal ganglia and showed that subcortical brain regions can be significantly predicted with genome-wide genotypes. This large-scale, long-running experiment demonstrated that our approaches are fault tolerant, and able to provide reliable, high-performance at large-scale for Big Data analysis. Finally, based on the overall validation with these real-life scenarios, we demonstrated that our data management contributions can indeed sustain and accelerate scientific discovery using the cloud.

High-Performance Inter-Site Transfers

An increasing number of Big Data applications are currently being ported on clouds to leverage the inherent elasticity and scalability of these multi-site infrastructures. In such a setting, it is critical to provide tools to support efficient sharing and dissemination of data across geographically-distributed sites. However, the existing cloud data management services lack mechanisms for dynamically coordinating transfers among different data centers and achieve reasonable QoS levels. Addressing this issue by providing high-performance, inter-site data transfer was a primary goal of this thesis.

To this end, we proposed several approaches, which cover the transfer from end to end. First, we devised a solution that separates the applications logic from the data sharing, increasing reliability and minimizing or controlling the intrusiveness of the transfer on the compute resources. Next, we proposed a strategy to allocate the nodes to be dedicated for managing the data in order to obtain performance levels similar to collocating the data locally within the compute nodes. Finally, we tackled the problem of the low inter-site bandwidth by proposing a multi-route transfer scheme that exploits the cloud architecture and the inherent network parallelism to aggregate extra bandwidth. These approaches significantly increase the performance over state-of-the-art alternatives, providing fast data transfers across sites and fulfilling the initial objective. Moreover, whether applied together or independently, these solutions optimize the management of data and the communication between application instances running across data centers, facilitating the migration to the cloud of the geographically distributed applications.

Monitoring Services to Track the Cloud Performance Status

Cloud infrastructures are exploited based on a multi-tenancy model, which leads to variations in the delivered performance of the compute nodes and the communication links. Monitoring and detecting such changes is particularly important for scientific applications which need predictable performance. This is achieved by collecting monitoring information about the cloud performance and by devising performance models to leverage these online observations. However, collecting monitoring samples is at odds with increasing the efficiency of applications, and consequently reducing the intrusiveness of the supporting services.

To address these issues we provide a lightweight monitoring service and a dynamic, generic sample-based model. The collected knowledge about the environment is made available to applications or is fed into higher-level management tools to leverage environment awareness. Hence, by monitoring the performance we are able to provision resources in advance, to remove the performance bottlenecks one-by-one, or to increase the end-to-end data transfer performance. This approach was leveraged in the context of multi-route, inter-site transfers. Additionally, the monitoring information collected by the services, is used also to calibrate the real-time streaming with respect to the cloud environment.

Improving Performance of Real-time Streaming through Environment-awareness

Rigid scheduling strategies are not the best fit for the dynamic environment of the cloud.The class of real-time streaming applications are particularly sensitive to the reactivity of the mechanisms of streaming the events to the environment, due to the potential small sizes of the events and the deadline constraints of the computation.

Therefore, we proposed a streaming model that adapts the batch size and the resource allocation, based on context parameters that are evaluated via monitoring. In this way, the streaming decision is controlled at runtime by minimizing the average latency according to the cloud streaming context. Based on this approach, we designed JetStream, which is able to provide high-performance streaming across cloud data centers. JetStream increases the transfer rates with orders of magnitude over individual or static event transfers strategies. This shows that leveraging adaptivity can significantly improve the performance of data

management in general, and provide efficient real-time communication across cloud sites in particular.

Customizable Trade-off between Cost and Performance

Finding value in large data sets is the main objective of Big Data. However, the costs users are willing to pay to retrieve this value and the performance levels they seek when performing this task are specific to each scenario. Therefore, an important focus of this work was to provide data management solutions which optimize for specific trade-offs between the cost and performance.

To this purpose, we modeled the relation between cost and performance for data transfers and designed a resource-provisioning strategy for customizable levels of performance according to cost constraints. This was applied both in the context of static and real-time data management. Furthermore, we investigated also the cost efficiency of the resources and provide automatic mechanisms for preventing resource wasting. The results showed that our solution is able to detect and adjust poor user configurations, which would otherwise lead to resource wasting, and that it can optimize data movements for optimal throughput, low latencies or best/custom ratios between price and delivered performance. Providing such customizable trade-off mechanisms for managing data is an important contribution for supporting Big Data processing, considering that the density of the value in the data sets is not uniform nor identical among different applications.

14.2 Perspectives

During this work, several choices were made regarding the directions and functionalities to be created to improve the cloud support for Big Data applications. Encouraged by the good results obtained, these contributions can now serve as a starting point for new research directions, complementing our work and extending the cloud data management ecosystem. The main perspectives opened by this thesis are presented next.

Data-Intensive Scientific Workflows on Geographically Distributed Cloud Sites

The global deployment of cloud data centers is enabling large-scale scientific workflows to improve performance and deliver fast responses. This unprecedented geographical distribution of computation is doubled by an increase in the scale of the data handled by such applications. Sustaining such a computation requires, alongside with high-performance geographically-distributed data management tools, processing engines which can provide efficient computation. Nevertheless, scheduling workflow processing across cloud data centers is not trivial as it requires adapting to the cloud environment and to the data layout to manage task placement. The main issues preventing this, but addressed in this thesis, were the limited knowledge available in the virtualized cloud user space and the inefficient data exchanges across sites. Therefore, the features brought by our approaches such as exposing the data layout to the workflow engine, optimizing data management according to the workflow access patterns, and enabling customizable cost/performance trade-offs, open

the perspectives of building a multi-site efficient workflow engine for scientific Big Data processing. This perspective is now the driving objective of the Z-CloudFlow project, sponsored by Microsoft and Inria within the context of the Microsoft Research - Inria Joint Center.

Stream Processing on Clouds

Stream data processing is becoming one of the most significant subclass of applications in the world of Big Data, with events being produced at growing rates by a variety of sources. Therefore, moving stream processing on clouds and scaling query processing is an open and hot issue nowadays. However, scaling the stream processing is a difficult and complex task due to the real-time nature of the analysis and the inherent latency-related deadline constraints. One of the most critical milestones concerning the stream latency is the management of the events, given that processing performance is in fact determined by the rate the data is provided to the compute engine. Nevertheless, by proposing a high-performance streaming middleware, we have made an important step towards the goal of enabling large-scale stream processing on clouds. Due to its generic architecture, JetStream can be bound to any event processing engine and applied to various scenarios. Withal, further improvements can be designed for particular scenarios, by customizing the streaming based on the application semantics or the query logic, to improve the performance of real-time applications on clouds.

Diversification of the Cloud Data Management Ecosystem

One of the key points emphasized and addressed in our work is the need for a richer functionality regarding data management on the clouds. Achieving this point is critical to improve the efficiency of the next-generation data centers, which are expected to provide mixed service models for accommodating the Big Data challenges. Therefore, we proposed several approaches that can serve as the foundation for developing new data-related functionalities. One interesting future direction is to extend the service processing layer, which we have built to capitalize the resources dedicated for the data management with new functionalities. A particularly promising aspect, considering the significant cost of inter-site data exchanges, is to provide online data deduplication techniques to eliminate redundant data transfers between data centers. The idea can be extended also to the context of streaming, by investigating trade-off options between latency and real-time compression mechanisms. Hence, building on the solutions proposed in this thesis, new services can be devised that would further enrich the cloud data services and better sustain large scale applications.

Cloud-provided Transfer as a Service and a Data Transfer Market

An important contribution of our work was to propose a dedicated cloud data transfer service that supports large-scale data dissemination across geographically distributed sites, advocating for a Transfer as a Service (TaaS) paradigm. We demonstrated significant benefits related to cost, performance and energy, that the adoption of such a service would bring for both users and cloud providers. Moreover, we devised a dynamic cost model scheme for the service usage, which enables the cloud providers to regulate and encourage data exchanges via a data transfer market. In particular, this opens the perspective of further studying new

cost models that allow users to bid on idle bandwidth and use it when their bid exceeds the current price, which varies in real-time based on supply and demand. Nevertheless, the main perspective opened by these contributions (Transfer as a Service and Data Transfer Market) is to migrate them to the cloud vendors.Hence, providing further evaluation studies covering diverse aspects, ranging from performance to marketing or network economics, is an important and impacting direction towards this goal.

Bibliography

[1] Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. "Aurora: A New Model and Architecture for Data Stream Management". In: *The VLDB Journal* 12.2 (Aug. 2003), pp. 120–139. ISSN: 1066-8888. DOI: 10.1007/s00778-003-0095-z. URL: http://dx.doi.org/10.1007/s00778-003-0095-z.

[2] Rohit Agarwal, Gideon Juve, and Ewa Deelman. "Peer-to-Peer Data Sharing for Scientific Workflows on Amazon EC2". In: *Proceedings of the 2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. SCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 82–89. ISBN: 978-0-7695-4956-9. DOI: 10.1109/SC.Companion.2012.23.

[3] W. Allcock. "GridFTP: Protocol Extensions to FTP for the Grid." In: *Global Grid ForumGFD-RP, 20* (2003).

[4] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. "The Globus Striped GridFTP Framework and Server". In: *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*. SC '05. Washington, DC, USA: IEEE Computer Society, 2005. ISBN: 1-59593-061-2.

[5] Beulah Kurian Alunkal. "Grid Eigen Trust a Framework for Computing Reputation in Grids". PhD thesis. Illinois Institute of Technology, 2003.

[6] *Amazon CloudFront*. http://aws.amazon.com/cloudfront/.

[7] *Amazon EBS*. http://aws.amazon.com/fr/ebs/.

[8] *Amazon Elastic MapReduce*. http://aws.amazon.com/elasticmapreduce/.

[9] *Amazon Web Services*. http://aws.amazon.com/.

[10] Ganesh Ananthanarayanan and Randy H. Katz. "Greening the Switch". In: *Proceedings of the 2008 Conference on Power Aware Computing and Systems*. HotPower'08. San Diego, California: USENIX Association.

[11] David P. Anderson. "BOINC: A System for Public-Resource Computing and Storage". In: *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*. GRID '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 4–10. ISBN: 0-7695-2256-4. DOI: 10.1109/GRID.2004.14. URL: http://dx.doi.org/10.1109/GRID.2004.14.

[12] M. Arrott, A. Clemesha, C. Farcas, E. Farcas, M. Meisinger, D. Raymer, D. LaBissoniere, and K. Keahey. "Cloud Provisioning Environment: Prototype Architecture and Technologies". In: *Ocean Observatories Initiative Kick Off Meeting* (September 2009).

[13] *ATLAS*. http://home.web.cern.ch/fr/about/experiments/atlas.

[14] *ATLAS Applications*. https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/PhysicsAnalysisWorkBookRel16D3PDAnalysisExample.

[15] *Azure Caching*. http://www.windowsazure.com/en-us/home/features/caching/.

[16] *Azure data centers*. http://blogs.msdn.com/b/briankel/archive/2014/04/10/building-your-dream-devops-dashboard-with-the-new-azure-preview-portal.aspx.

[17] *Azure Drives*. http://msdn.microsoft.com/en-us/library/windowsazure/jj156162.aspx.

[18] *Azure Failure Incident*. http://azure.microsoft.com/blog/2012/03/09/summary-of-windows-azure-service-disruption-on-feb-29th-2012/.

[19] *Azure Priceing*. https://www.windowsazure.com/en-us/pricing/details/.

[20] *Azure Succesful Stories*. http://www.windowsazure.com/en-us/case-studies/archive/.

[21] Jan Balewski, Jerome Lauret, Doug Olson, Iwona Sakrejda, Dmitry Arkhipkin, John Bresnahan, Kate Keahey, Jeff Porter, Justin Stevens, and Matt Walker. "Offloading peak processing to virtual farm by STAR experiment at RHIC". In: *Journal of Physics: Conference Series* (2012).

[22] Cagri Balkesen, Nihal Dindar, Matthias Wetter, and Nesime Tatbul. "RIP: Run-based Intra-query Parallelism for Scalable Complex Event Processing". In: *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*. DEBS '13. Arlington, Texas, USA: ACM, 2013, pp. 3–14. ISBN: 978-1-4503-1758-0. DOI: 10.1145/2488222.2488257. URL: http://doi.acm.org/10.1145/2488222.2488257.

[23] António Baptista, Bill Howe, Juliana Freire, David Maier, and Cláudio T. Silva. "Scientific Exploration in the Era of Ocean Observatories". In: *Computing in Science and Engg.* 10.3 (May 2008), pp. 53–58. ISSN: 1521-9615. DOI: 10.1109/MCSE.2008.83. URL: http://dx.doi.org/10.1109/MCSE.2008.83.

[24] Gordon Bell, Tony Hey, and Alex Szalay. "Beyond the Data Deluge". In: *Science* 323.5919 (Mar. 2009), pp. 1297–1298. ISSN: 1095-9203. DOI: 10.1126/science.1170411. URL: http://dx.doi.org/10.1126/science.1170411.

[25] *Big Data - What it is and why it matters*. http://www.sas.com/en_us/insights/big-data/what-is-big-data.html.

[26] Bahar Biller and Barry L. Nelson. "Modeling and Generating Multivariate Time-series Input Processes Using a Vector Autoregressive Technique". In: *ACM Trans. Model. Comput. Simul.* 13.3 (July 2003), pp. 211–237. ISSN: 1049-3301. DOI: 10.1145/937332.937333. URL: http://doi.acm.org/10.1145/937332.937333.

[27] Tony Bishop. "Data Center 2.0 A Roadmap for Data Center Transformation". In: *White Paper*. http://www.io.com/white-papers/data-center-2-roadmap-for-data-center-transformation/, 2013.

[28] *BitTorrent library*. http://www.mono-project.com/MonoTorrent.

[29] *Bluewaters supercomputer applications*. http://www.ncsa.illinois.edu/enabling/vis.

[30] B. Bond. *Best Practices for Developing on Window Azure*. http://azurescope.cloudapp.net/BestPractices/.

[31] Irina Botan, Gustavo Alonso, Peter M. Fischer, Donald Kossmann, and Nesime Tatbul. "Flexible and Scalable Storage Management for Data-intensive Stream Processing". In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. EDBT '09. Saint Petersburg, Russia: ACM, 2009, pp. 934–945. ISBN: 978-1-60558-422-5. DOI: 10.1145/1516360.1516467. URL: http://doi.acm.org/10.1145/1516360.1516467.

[32] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. "HaLoop: efficient iterative data processing on large clusters". In: *Proc. VLDB Endow.* 3 (1-2 2010), pp. 285–296. ISSN: 2150-8097.

[33] Krishnaveni Budati, Jason Sonnek, Abhishek Chandra, and Jon Weissman. "Ridge: Combining Reliability and Performance in Open Grid Platforms". In: *Proceedings of the 16th International Symposium on High Performance Distributed Computing*. HPDC '07. Monterey, California, USA: ACM, 2007, pp. 55–64. ISBN: 978-1-59593-673-8. DOI: 10.1145/1272366.1272374. URL: http://doi.acm.org/10.1145/1272366.1272374.

[34] Rajkumar Buyya. "Market-Oriented Cloud Computing: Vision, Hype, and Reality of Delivering Computing As the 5th Utility". In: *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*. CCGRID '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–. ISBN: 978-0-7695-3622-4. DOI: 10.1109/CCGRID.2009.97. URL: http://dx.doi.org/10.1109/CCGRID.2009.97.

[35] Brad Calder, Ju Wang, Aaron Ogus, Niranjan Nilakantan, Arild Skjolsvold, Sam McKelvie, Yikang Xu, Shashwat Srivastav, Jiesheng Wu, Huseyin Simitci, Jaidev Haridas, Chakravarthy Uddaraju, Hemal Khatri, Andrew Edwards, Vaman Bedekar, Shane Mainali, Rafay Abbasi, Arpit Agarwal, Mian Fahim ul Haq, Muhammad Ikram ul Haq, Deepali Bhardwaj, Sowmya Dayanand, Anitha Adusumilli, Marvin McNett, Sriram Sankaran, Kavitha Manivannan, and Leonidas Rigas. "Windows Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: ACM, 2011, pp. 143–157. ISBN: 978-1-4503-0977-6. DOI: 10.1145/2043556.2043571. URL: http://doi.acm.org/10.1145/2043556.2043571.

[36] Ümit V. Çatalyürek, Kamer Kaya, and Bora Uçar. "Integrated data placement and task assignment for scientific workflows in clouds". In: *Proceedings of the fourth international workshop on Data-intensive distributed computing*. DIDC '11. San Jose, California, USA: ACM, 2011, pp. 45–54. ISBN: 978-1-4503-0704-8.

[37] *CERN Alice*. http://alimonitor.cern.ch/map.jsp.

[38] *CERN Large Hadron Collider*. http://home.web.cern.ch/topics/large-hadron-collider.

[39] Badrish Chandramouli, Jonathan Goldstein, Roger Barga, Mirek Riedewald, and Ivo Santos. "Accurate Latency Estimation in a Distributed Event Processing System". In: *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*. ICDE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 255–266. ISBN: 978-1-4244-8959-6. DOI: 10.1109/ICDE.2011.5767926. URL: http://dx.doi.org/10.1109/ICDE.2011.5767926.

[40] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. "Bigtable: A Distributed Storage System for Structured Data". In: *ACM Trans. Comput. Syst.* 26.2 (June 2008), 4:1–4:26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816. URL: http://doi.acm.org/10.1145/1365815.1365816.

[41] Huanhuan Chen, Fengzhen Tang, Peter Tino, and Xin Yao. "Model-based Kernel for Efficient Time Series Analysis". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: ACM, 2013, pp. 392–400. ISBN: 978-1-4503-2174-7. DOI: 10.1145/2487575.2487700. URL: http://doi.acm.org/10.1145/2487575.2487700.

[42] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. "Virtual network embedding through topology-aware node ranking". In: *SIGCOMM Comput. Commun. Rev.* 41.2 (Apr. 2011).

[43] *Cloud Computing and High-Energy Particle Physics: How ATLAS Experiment at CERN Uses Google Compute Engine in the Search for New Physics at LHC*. https://developers.google.com/events/io/sessions/333315382.

[44] Brian F. Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. "PNUTS: Yahoo!'s Hosted Data Serving Platform". In: *Proc. VLDB Endow.* 1.2 (Aug. 2008), pp. 1277–1288. ISSN: 2150-8097. DOI: 10.14778/1454159.1454167. URL: http://dx.doi.org/10.14778/1454159.1454167.

[45] Benoit Da Mota, Vincent Frouin, Edouard Duchesnay, Soizic Laguitton, Gaël Varoquaux, Jean-Baptiste Poline, and Bertrand Thirion. "A fast computational framework for genome-wide association studies with neuroimaging data". In: *20th International Conference on Computational Statistics (COMPSTAT 2012)*. Limassol, Chypre, 2012. URL: http://hal.inria.fr/hal-00720265.

[46] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492.

[47] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. "Dynamo: Amazon's Highly Available Key-value Store". In: *SIGOPS Oper. Syst. Rev.* 41.6 (Oct. 2007), pp. 205–220. ISSN: 0163-5980. DOI: 10.1145/1323293.1294281. URL: http://doi.acm.org/10.1145/1323293.1294281.

[48] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia Laity, Joseph C. Jacob, and Daniel S. Katz. "Pegasus: A Framework for Mapping Complex Scientific Workflows Onto Distributed Systems". In: *Sci. Program.* 13.3 (July 2005), pp. 219–237. ISSN: 1058-9244. URL: http://dl.acm.org/citation.cfm?id=1239649.1239653.

[49] Gianluca Demartini, Beth Trushkowsky, Tim Kraska, and Michael J. Franklin. "CrowdQ: Crowdsourced Query Understanding." In: *CIDR*. www.cidrdb.org, 2003.

[50] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, Marc Snir, and Leigh Orf. "Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O". In: *Proceedings of the 2012 IEEE International Conference on Cluster Computing*. CLUSTER '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 155–163. ISBN: 978-0-7695-4807-4. DOI: 10.1109/CLUSTER.2012.26. URL: http://dx.doi.org/10.1109/CLUSTER.2012.26.

[51] Edd Dumbill. *What is Big Data?* Tech. rep. Oreilly Radar, 2012. URL: http://radar.oreilly.com/2012/01/what-is-big-data.html.

[52] Nigel Edwards, Mark Watkins, Matt Gates, Alistair Coles, Eric Deliot, Aled Edwards, Anna Fischer, Patrick Goldsack, Tom Hancock, Donagh McCabe, Tim Reddin, JP Sullivan, Peter Toft, and Lawrence Wilcock. "High-speed Storage Nodes for the Cloud". In: *Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing*. UCC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 25–32. ISBN: 978-0-7695-4592-9. DOI: 10.1109/UCC.2011.14. URL: http://dx.doi.org/10.1109/UCC.2011.14.

[53] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. "Twister: a runtime for iterative MapReduce". In: *HPDC '10*. Chicago, Illinois, 2010, pp. 810–818. ISBN: 978-1-60558-942-8.

[54] *Eucalyptus*. https://www.eucalyptus.com/.

[55] Pei Fan, Zhenbang Chen, Ji Wang1, Zibin Zheng, and Michael R. Lyu. "Scientific application deployment on Cloud: A Topology-Aware Method". In: *Concurrency and Computattion: Practice and Experience* (2012).

[56] Pei Fan, Zhenbang Chen, Ji Wang, Zibin Zheng, and Michael R. Lyu. "Topology-Aware Deployment of Scientific Applications in Cloud Computing". In: *IEEE CLOUD*. 2012.

[57] Eugen Feller, Lavanya Ramakrishnan, and Christine Morin. "On the Performance and Energy Efficiency of Hadoop Deployment Models". In: *The IEEE International Conference on Big Data 2013 (IEEE BigData 2013)*. Grid'5000 Grid'5000. Santa Clara, États-Unis, Oct. 2013.

[58] Ian Foster, Rajkumar Kettimuthu, Stuart Martin, Steve Tuecke, Daniel Milroy, Brock Palen, Thomas Hauser, and Jazcek Braden. "Campus Bridging Made Easy via Globus Services". In: *Proceedings of the 1st Conference of the Extreme Science and Engineering Discovery Environment: Bridging from the eXtreme to the Campus and Beyond*. XSEDE '12. Chicago, Illinois: ACM, 2012, 50:1–50:8. ISBN: 978-1-4503-1602-6. DOI: 10.1145/2335755.2335847.

[59] Ian Foster, Ann Chervenak, Dan Gunter, Kate Keahey, Ravi Madduri, and Raj Kettimuthu. "Enabling PETASCALE Data Movement and Analysis". In: *Scidac Review* (Winter 2009). URL: http://www.scidacreview.org/0905/pdf/cedps.pdf.

[60] Michael J. Franklin, Beth Trushkowsky, Purnamrita Sarkar, and Tim Kraska. "Crowdsourced Enumeration Queries". In: *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*. ICDE '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 673–684. ISBN: 978-1-4673-4909-3. DOI: 10.1109/ICDE.2013.6544865. URL: http://dx.doi.org/10.1109/ICDE.2013.6544865.

[61] *FTP library*. http://www.codeproject.com/Articles/380769.

[62] *FutureGrid*. https://portal.futuregrid.org/.

[63] Sahan Gamage, Ramana Rao Kompella, Dongyan Xu, and Ardalan Kangarlou. "Protocol Responsibility Offloading to Improve TCP Throughput in Virtualized Environments". In: *ACM Trans. Comput. Syst.* 31.3 (Aug. 2013), 7:1–7:34. ISSN: 0734-2071. DOI: 10.1145/2491463.

[64] John Gantz and David Reinsel. *THE DIGITAL UNIVERSE IN 2020: Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East*. Tech. rep. Internet Data Center(IDC), 2012. URL: http://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf.

[65] Bugra Gedik, Henrique Andrade, Kun-Lung Wu, Philip S. Yu, and Myungcheol Doo. "SPADE: The System s Declarative Stream Processing Engine". In: *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*. SIGMOD '08. Vancouver, Canada: ACM, 2008, pp. 1123–1134. ISBN: 978-1-60558-102-6. DOI: 10.1145/1376616.1376729. URL: http://doi.acm.org/10.1145/1376616.1376729.

[66] *Generic Worker*. http://www.venus-c.eu/Pages/Home.aspx.

[67] Devarshi Ghoshal, Richard Shane Canon, and Lavanya Ramakrishnan. "I/O Performance of Virtualized Cloud Environments". In: *Proceedings of the Second International Workshop on Data Intensive Computing in the Clouds*. DataCloud-SC '11. Seattle, Washington, USA: ACM, 2011, pp. 71–80. ISBN: 978-1-4503-1144-1. DOI: 10.1145/2087522.2087535. URL: http://doi.acm.org/10.1145/2087522.2087535.

[68] Lukasz Golab and M. Tamer Özsu. "Issues in Data Stream Management". In: *SIGMOD Rec.* 32.2 (June 2003), pp. 5–14. ISSN: 0163-5808. DOI: 10.1145/776985.776986. URL: http://doi.acm.org/10.1145/776985.776986.

[69] L. A. Bautista Gomez and F. Cappello. "Improving Floating Point Compression through Binary Masks". In: *IEEE BigData 2013*. Santa Barbara, California, 2013.

[70] A Gomez-Iglesias, AT. Ernst, and G. Singh. "Scalable Multi Swarm-Based Algorithms with Lagrangian Relaxation for Constrained Problems". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. 2013, pp. 1073–1080. DOI: 10.1109/TrustCom.2013.241.

[71] *Google Apps Engine*. https://cloud.google.com/.

[72] Jim Gray. *A Transformed Scientific Method*. http://research.microsoft.com/en-us/um/people/gray/talks/NRC-CSTB_eScience.ppt. 2007.

[73] Jim Gray and Alex Szalay. "Science In An Exponential World". In: *Nature* 440.23 (2006). URL: http://research.microsoft.com/apps/pubs/default.aspx?id=64526.

[74] Albert Greenberg, James Hamilton, David A. Maltz, and Parveen Patel. "The Cost of a Cloud: Research Problems in Data Center Networks". In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2008), pp. 68–73. ISSN: 0146-4833. DOI: 10.1145/1496091.1496103.

[75] Yunhong Gu and Robert L. Grossman. "Sector and Sphere: The Design and Implementation of a High-Performance Data Cloud". In: *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 367.1897 (2009), pp. 2429–2445. ISSN: 1364503X. URL: http://www.jstor.org/stable/40485591.

[76] Thilina Gunarathne, Tak-Lon Wu, Jong Youl Choi, Seung-Hee Bae, and Judy Qiu. "Cloud Computing Paradigms for Pleasingly Parallel Biomedical Applications". In: *Concurr. Comput. : Pract. Exper.* 23.17 (Dec. 2011), pp. 2338–2354. ISSN: 1532-0626. DOI: 10.1002/cpe.1780. URL: http://dx.doi.org/10.1002/cpe.1780.

[77] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, and Geoffrey Fox. "MapReduce in the Clouds for Science". In: *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*. CLOUDCOM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 565–572. ISBN: 978-0-7695-4302-4. DOI: 10.1109/CloudCom.2010.107. URL: http://dx.doi.org/10.1109/CloudCom.2010.107.

[78] Thilina Gunarathne, Bingjing Zhang, Tak-Lon Wu, and Judy Qiu. "Scalable Parallel Computing on Clouds Using Twister4Azure Iterative MapReduce". In: *Future Generation Computer Systems.* (2012). ISSN: 0167-739X.

[79] Abhishek Gupta, Ozgur D. Sahin, Divyakant Agrawal, and Amr El Abbadi. "Meghdoot: Content-based Publish/Subscribe over P2P Networks". In: *Proceedings of the 5th ACM/IFIP/USENIX International Conference on Middleware*. Middleware '04. Toronto, Canada: Springer-Verlag New York, Inc., 2004, pp. 254–273. ISBN: 3-540-23428-4. URL: http://dl.acm.org/citation.cfm?id=1045658.1045677.

[80] T. J. Hacker, B. D. Noble, and B. D. Athey. "Adaptive data block scheduling for parallel TCP streams". In: *Proceedings of the High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium*. HPDC '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 265–275. ISBN: 0-7803-9037-7.

[81] Ibrahim F. Haddad. "PVFS: A Parallel Virtual File System for Linux Clusters". In: *Linux J.* 2000.80es (Nov. 2000). ISSN: 1075-3583. URL: http://dl.acm.org/citation.cfm?id=364352.364654.

[82] *Hadoop*. http://hadoop.apache.org/.

[83] P. Hande, Mung Chiang, R. Calderbank, and S. Rangan. "Network Pricing and Rate Allocation with Content Provider Participation". In: *INFOCOM 2009, IEEE*. 2009, pp. 990–998. DOI: 10.1109/INFCOM.2009.5062010.

[84] M. Hayes and S. Shah. "Hourglass: A library for incremental processing on Hadoop". In: *Big Data, 2013 IEEE International Conference on*. 2013, pp. 742–752. DOI: 10.1109/BigData.2013.6691647.

[85] *HDFS*. http://hadoop.apache.org/hdfs/.

[86] *HDInsight (Hadoop on Azure)*. https://www.hadooponazure.com/.

[87] Bingsheng He, Mao Yang, Zhenyu Guo, Rishan Chen, Bing Su, Wei Lin, and Lidong Zhou. "Comet: Batched Stream Processing for Data Intensive Distributed Computing". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 63–74. ISBN: 978-1-4503-0036-0. DOI: 10.1145/1807128.1807139. URL: http://doi.acm.org/10.1145/1807128.1807139.

[88] *HeavyLoad*. http://www.jam-software.com/heavyload/.

[89] Tony Hey, Stewart Tansley, and Kristin M. Tolle, eds. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, 2009. ISBN: 978-0982544204.

[90] Tony Hey and Anne E Trefethen. "Cyberinfrastructure for e-Science". In: *Science* 308.5723 (2005), pp. 817–821.

[91] H.Hiden, S. Woodman, P.Watson, and J.Cała. "Developing Cloud Applications using the e-Science Central Platform." In: *Proceedings of Royal Society A*. Vol. 371. 1983. Dec. 2012. DOI: 10.1098/rsta.2012.0085.

[92] H. Hiden, S. Woodman, P. Watson, M. Catt, M. Trenell, and S. Zhang. "Improving the scalability of movement monitoring workflows: An architecture for the integration of the Hadoop File System into e-Science Central." In: *Proceedings of 1st Conference on Digital Research*. Oxford, UK, 2012.

[93] Zach Hill, Jie Li, Ming Mao, Arkaitz Ruiz-Alvarez, and Marty Humphrey. "Early observations on the performance of Windows Azure". In: *Sci. Program.* 19.2-3 (Apr. 2011), pp. 121–132. ISSN: 1058-9244.

[94] *Hive*. http://hive.apache.org/.

[95] *IBM big data and information management*. http://www-01.ibm.com/software/data/bigdata/.

[96] Roger Immich, Eduardo Cerqueira, and Marilia Curado. "Adaptive Video-aware FEC-based Mechanism with Unequal Error Protection Scheme". In: *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. SAC '13. Coimbra, Portugal: ACM, 2013, pp. 981–988. ISBN: 978-1-4503-1656-9. DOI: 10.1145/2480362.2480550. URL: http://doi.acm.org/10.1145/2480362.2480550.

[97] *Information facts*. http://searchstorage.techtarget.com/definition/How-many-bytes-for.

[98] *Resilin: Elastic MapReduce over Multiple Clouds*. IEEE Computer Society, 2013. ISBN: 978-1-4673-6465-2.

[99] *Iperf*. http://iperf.fr/.

[100] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. "Dryad: distributed data-parallel programs from sequential building blocks". In: *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. EuroSys '07. Lisbon, Portugal: ACM, 2007, pp. 59–72. ISBN: 978-1-59593-636-3. DOI: 10.1145/1272996.1273005.

[101] A. Ishii and T. Suzumura. "Elastic Stream Computing with Clouds". In: *Cloud Computing (CLOUD), 2011 IEEE International Conference on*. 2011, pp. 195–202. DOI: 10.1109/CLOUD.2011.11.

[102] Keith R. Jackson, Lavanya Ramakrishnan, Karl J. Runge, and Rollin C. Thomas. "Seeking supernovae in the clouds: a performance study". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, 2010, pp. 421–429. ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851538.

[103] Gideon Juve, Ewa Deelman, G. Bruce Berriman, Benjamin P. Berman, and Philip Maechling. "An Evaluation of the Cost and Performance of Scientific Workflows on Amazon EC2". In: *J. Grid Comput.* 10.1 (Mar. 2012), pp. 5–21. ISSN: 1570-7873. DOI: 10.1007/s10723-012-9207-6.

[104] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, and Bruce Berriman. "Scientific workflow applications on Amazon EC2". In: *In Cloud Computing Workshop in Conjunction with e-Science*. IEEE, 2009.

[105] Wolfgang Kabsch. "A solution for the best rotation to relate two sets of vectors". In: *Acta Crystallographica A , 32:922–923*. 1976.

[106] V. Kalavri and V. Vlassov. "MapReduce: Limitations, Optimizations and Open Issues". In: *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*. 2013, pp. 1031–1038. DOI: 10.1109/TrustCom.2013.126.

[107] Katarzyna Keahey, Maurício Tsugawa, Andréa Matsunaga, and José A.B. Fortes. "Sky Computing". In: *Cloud Computing* (2009).

[108] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, and J. Saltz. "A dynamic scheduling approach for coordinated wide-area data transfers using GridFTP". In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. 2008, pp. 1–12.

[109] Gaurav Khanna, Umit Catalyurek, Tahsin Kurc, Rajkumar Kettimuthu, P. Sadayappan, Ian Foster, and Joel Saltz. "Using overlays for efficient data transfer over shared wide-area networks". In: *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. SC '08. Austin, Texas: IEEE Press, 2008, 47:1–47:12. ISBN: 978-1-4244-2835-9. URL: http://dl.acm.org/citation.cfm?id=1413370.1413418.

[110] Romeo Kienzler, Remy Bruggmann, Anand Ranganathan, and Nesime Tatbul. "Stream as You Go: The Case for Incremental Data Access and Processing in the Cloud". In: *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering Workshops*. ICDEW '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 159–166. ISBN: 978-0-7695-4748-0.

[111] Tevfik Kosar and Miron Livny. "A Framework for Reliable and Efficient Data Placement in Distributed Computing Systems". In: *J. Parallel Distrib. Comput.* 65.10 (Oct. 2005), pp. 1146–1157. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2005.04.019. URL: http://dx.doi.org/10.1016/j.jpdc.2005.04.019.

[112] Tevfik Kosar, Engin Arslan, Brandon Ross, and Bing Zhang. "StorkCloud: Data Transfer Scheduling and Optimization As a Service". In: *Proceedings of the 4th ACM Workshop on Scientific Cloud Computing*. Science Cloud '13. New York, New York, USA: ACM, 2013, pp. 29–36. ISBN: 978-1-4503-1979-9. DOI: 10.1145/2465848.2465855.

[113] Avinash Lakshman and Prashant Malik. "Cassandra: A Decentralized Structured Storage System". In: *SIGOPS Oper. Syst. Rev.* 44.2 (Apr. 2010), pp. 35–40. ISSN: 0163-5980. DOI: 10.1145/1773912.1773922. URL: http://doi.acm.org/10.1145/1773912.1773922.

[114] Chhagan Lal, Vijay Laxmi, and Manoj Singh Gaur. "A Rate Adaptive and Multipath Routing Protocol to Support Video Streaming in MANETs". In: *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ICACCI '12. Chennai, India: ACM, 2012, pp. 262–268. ISBN: 978-1-4503-1196-0. DOI: 10.1145/2345396.2345440. URL: http://doi.acm.org/10.1145/2345396.2345440.

[115] Doug Laney. *3D Data Management: Controlling Data Volume, Veracity, and Variety*. Tech. rep. Meta Group, 2011. URL: http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data-Management-Controlling-Data-Volume-Velocity-and-Variety.pdf.

[116] Nikolaos Laoutaris, Michael Sirivianos, Xiaoyuan Yang, and Pablo Rodriguez. "Inter-datacenter Bulk Transfers with Netstitcher". In: *Proceedings of the ACM SIGCOMM 2011 Conference*. SIGCOMM '11. Toronto, Ontario, Canada: ACM, 2011, pp. 74–85. ISBN: 978-1-4503-0797-0. DOI: 10.1145/2018436.2018446.

[117] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, C. Dobre, A. Muraru, A. Costan, M. Dediu, and C. Stratan. "MonALISA: An agent based, dynamic service system to monitor, control and optimize distributed systems". In: *Computer Physics Communications* 180.12 (2009). 40 {YEARS} {OF} CPC: A celebratory issue focused on quality software for high performance, grid and novel computing architectures, pp. 2472 –2498. ISSN: 0010-4655. DOI: http://dx.doi.org/10.1016/j.cpc.2009.08.003. URL: http://www.sciencedirect.com/science/article/pii/S0010465509002410.

[118] Wantao Liu, Brian Tieman, Rajkumar Kettimuthu, and Ian Foster. "A data transfer framework for large-scale science experiments". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, 2010, pp. 717–724. ISBN: 978-1-60558-942-8.

[119] Wantao Liu, Brian Tieman, Rajkumar Kettimuthu, and Ian Foster. "A data transfer framework for large-scale science experiments". In: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. HPDC '10. Chicago, Illinois: ACM, 2010, pp. 717–724. ISBN: 978-1-60558-942-8. DOI: 10.1145/1851476.1851582. URL: http://doi.acm.org/10.1145/1851476.1851582.

[120] Simon Loesing, Martin Hentschel, Tim Kraska, and Donald Kossmann. "Stormy: An Elastic and Highly Available Streaming Service in the Cloud". In: *Proceedings of the 2012 Joint EDBT/ICDT Workshops*. EDBT-ICDT '12. Berlin, Germany: ACM, 2012, pp. 55–60. ISBN: 978-1-4503-1143-4. DOI: 10.1145/2320765.2320789. URL: http://doi.acm.org/10.1145/2320765.2320789.

[121] Yuan Luo and Beth Plale. "Hierarchical MapReduce Programming Model and Scheduling Algorithms". In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (Ccgrid 2012)*. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 769–774. ISBN: 978-0-7695-4691-9. DOI: 10.1109/CCGrid.2012.132. URL: http://dx.doi.org/10.1109/CCGrid.2012.132.

[122] Kasper Grud Skat Madsen, Li Su, and Yongluan Zhou. "Grand Challenge: MapReduce-style Processing of Fast Sensor Data". In: *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*. DEBS '13. Arlington, Texas, USA: ACM, 2013, pp. 313–318. ISBN: 978-1-4503-1758-0. DOI: 10.1145/2488222.2488281. URL: http://doi.acm.org/10.1145/2488222.2488281.

[123] Shiwen Mao, Xiaolin Cheng, Y.Thomas Hou, and HanifD. Sherali. "Multiple Description Video Multicast in Wireless Ad Hoc Networks". English. In: *Mobile Networks and Applications* 11.1 (2006), pp. 63–73. ISSN: 1383-469X. DOI: 10.1007/s11036-005-4461-5. URL: http://dx.doi.org/10.1007/s11036-005-4461-5.

[124] *MEF*. http://msdn.microsoft.com/en-us/library/dd460648.aspx.

[125] *Microsoft Azure*. http://azure.microsoft.com/en-us/.

[126] *Microsoft Data Initiative*. http://research.microsoft.com/en-us/projects/azure/cdi.aspx.

[127] *Microsoft StreamInsight*. http://technet.microsoft.com/en-us/library/ee362541(v=sql.111).aspx.

[128] Diana Moise, Gabriel Antoniu, and Luc Bougé. "On-the-fly Task Execution for Speeding Up Pipelined Mapreduce". In: *Proceedings of the 18th International Conference on Parallel Processing*. Euro-Par'12. Rhodes Island, Greece: Springer-Verlag, 2012, pp. 526–537. ISBN: 978-3-642-32819-0. DOI: 10.1007/978-3-642-32820-6_52. URL: http://dx.doi.org/10.1007/978-3-642-32820-6_52.

[129] Henry M. Monti, Ali R. Butt, and Sudharshan S. Vazhkudai. "CATCH: A Cloud-Based Adaptive Data Transfer Service for HPC". In: *Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium*. IPDPS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1242–1253. ISBN: 978-0-7695-4385-7. DOI: 10.1109/IPDPS.2011.118. URL: http://dx.doi.org/10.1109/IPDPS.2011.118.

[130] Bogdan Nicolae, Pierre Riteau, and Kate Keahey. "Bursting the Cloud Data Bubble: Towards Transparent Storage Elasticity in IaaS Clouds." In: *In IPDPS '14: Proc. 28th IEEE International Parallel and Distributed Processing Symposium*. Phoenix, Arizona, 2014.

[131] Bogdan Nicolae, Gabriel Antoniu, Luc Bougé, Diana Moise, and Ra Carpen-amarie. "Blobseer: Next-generation data management for large scale infrastructures". In: *J. Parallel Distrib. Comput* (2011), pp. 169–184.

[132] Bogdan Nicolae, Gabriel Antoniu, Luc Bougé, Diana Moise, and Ra Carpen-amarie. "Blobseer: Next-generation data management for large scale infrastructures". In: *J. Parallel Distrib. Comput* (2011), pp. 169–184.

[133] Bogdan Nicolae, John Bresnahan, Kate Keahey, and Gabriel Antoniu. "Going Back and Forth: Efficient Multideployment and Multisnapshotting on Clouds". In: *Proceedings of the 20th International Symposium on High Performance Distributed Computing*. HPDC '11. San Jose, California, USA: ACM, 2011, pp. 147–158. ISBN: 978-1-4503-0552-5. DOI: 10.1145/1996130.1996152. URL: http://doi.acm.org/10.1145/1996130.1996152.

[134] *Nimbus*. http://www.nimbusproject.org/.

[135] *Nimbus*. http://nimbusproject.org/.

[136] K. Hiraga O. Tatebe and N. Soda. "Gfarm grid file system". In: *New Generation Computing: 257-275*. Vol. 28. 3. 2010, pp. 257 –275.

[137] *Ocean Observatory Initiative*. http://oceanobservatories.org/.

[138] *OpenNebula*. http://www.opennebula.org/.

[139] *OpenStack*. http://www.openstack.org/.

[140] *OpenStack Grizzly*. http://openstack.org/software/grizzly/.

[141] Anand Padmanabhan, Shaowen Wang, Guofeng Cao, Myunghwa Hwang, Yanli Zhao, Zhenhua Zhang, and Yizhao Gao. "FluMapper: an interactive CyberGIS environment for massive location-based social media data analysis". In: *Proceedings of the Conference on Extreme Science and Engineering Discovery Environment: Gateway to Discovery*. XSEDE '13. San Diego, California: ACM, 2013, 33:1–33:2. ISBN: 978-1-4503-2170-9.

[142] *Pan-STARRS: Panoramic Survey Telescope and Rapid Response System*. http://pan-starrs.ifa.hawaii.edu.

[143] A. M. Parker. *Understanding the Universe*. Tech. rep. Towards 2020 Science, Microsoft Corporation, 2006. URL: http://research.microsoft.com/towards2020science/background_overview.htm.

[144] Andrew Pavlo, Erik Paulson, Alexander Rasin, Daniel J. Abadi, David J. DeWitt, Samuel Madden, and Michael Stonebraker. "A Comparison of Approaches to Large-scale Data Analysis". In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. SIGMOD '09. Providence, Rhode Island, USA: ACM, 2009, pp. 165–178. ISBN: 978-1-60558-551-2. DOI: 10.1145/1559845.1559865. URL: http://doi.acm.org/10.1145/1559845.1559865.

[145] *Pig*. http://pig.apache.org/.

[146] Jean-Baptiste Poline, Christophe Lalanne, Arthur Tenenhaus, Edouard Duchesnay, Bertrand Thirion, and Vincent Frouin. "Imaging genetics: bio-informatics and biostatistics challenges". In: *19th International Conference on Computational Statistics*. Paris, France, Aug. 2010. URL: http://hal.inria.fr/inria-00523236.

[147] Krishna P.N. Puttaswamy, Thyaga Nandagopal, and Murali Kodialam. "Frugal Storage for Cloud File Systems". In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. Bern, Switzerland: ACM, 2012, pp. 71–84. ISBN: 978-1-4503-1223-3. DOI: 10.1145/2168836.2168845.

[148] Costin Raiciu, Christopher Pluntke, Sebastien Barre, Adam Greenhalgh, Damon Wischik, and Mark Handley. "Data center networking with multipath TCP". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 10:1–10:6. ISBN: 978-1-4503-0409-2.

[149] S. Sakr, A. Liu, D.M. Batista, and M. Alomari. "A Survey of Large Scale Data Management Approaches in Cloud Environments". In: *Communications Surveys Tutorials, IEEE* 13.3 (2011), pp. 311–336. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.032211.00087.

[150] Sherif Sakr, Anna Liu, Daniel M. Batista, and Mohammad Alomari. "A Survey of Large Scale Data Management Approaches in Cloud Environments". In: *IEEE Communications Surveys and Tutorials* 13.3 (2011), pp. 311–336.

[151] Srinivas Shakkottai and R. Srikant. "Economics of Network Pricing with Multiple ISPs". In: *IEEE/ACM Trans. Netw.* 14.6 (Dec. 2006), pp. 1233–1245. ISSN: 1063-6692. DOI: 10.1109/TNET.2006.886393.

[152] Mohamed A. Sharaf, Panos K. Chrysanthis, and Alexandros Labrinidis. "Tuning QoD in Stream Processing Engines". In: *Proceedings of the Twenty-First Australasian Conference on Database Technologies - Volume 104*. ADC '10. Brisbane, Australia: Australian Computer Society, Inc., 2010, pp. 103–112. ISBN: 978-1-920682-85-9. URL: http://dl.acm.org/citation.cfm?id=1862242.1862257.

[153] Yogesh Simmhan, Catharine van Ingen, Girish Subramanian, and Jie Li. "Bridging the Gap between Desktop and the Cloud for eScience Applications". In: *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*. CLOUD '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 474–481. ISBN: 978-0-7695-4130-3. DOI: 10.1109/CLOUD.2010.72. URL: http://dx.doi.org/10.1109/CLOUD.2010.72.

[154] Aameek Singh, Mudhakar Srivatsa, and Ling Liu. "Search-as-a-service: Outsourced Search over Outsourced Storage". In: *ACM Trans. Web* 3.4 (Sept. 2009), 13:1–13:33. ISSN: 1559-1131. DOI: 10.1145/1594173.1594175. URL: http://doi.acm.org/10.1145/1594173.1594175.

[155] *Spark*. https://spark.apache.org/.

[156] *Square Kilometre Array Telescope*. http://www.skatelescope.org/.

[157] Ion Stoica, Robert Morris, David Liben-Nowell, David R. Karger, M. Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications". In: *IEEE/ACM Trans. Netw.* 11.1 (Feb. 2003), pp. 17–32. ISSN: 1063-6692. DOI: 10.1109/TNET.2002.808407. URL: http://dx.doi.org/10.1109/TNET.2002.808407.

[158] AS. Szalay, G. Bell, J. VandenBerg, A Wonders, R. Burns, Dan Fay, J. Heasley, T. Hey, M. Nieto-Santisteban, A Thakar, C. van Ingen, and R. Wilton. "GrayWulf: Scalable Clustered Architecture for Data Intensive Computing". In: *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*. 2009, pp. 1–10. DOI: 10.1109/HICSS.2009.234.

[159] Muhammad Adnan Tariq, Boris Koldehofe, and Kurt Rothermel. "Efficient Content-based Routing with Network Topology Inference". In: *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*. DEBS '13. Arlington, Texas, USA: ACM, 2013, pp. 51–62. ISBN: 978-1-4503-1758-0. DOI: 10.1145/2488222.2488262. URL: http://doi.acm.org/10.1145/2488222.2488262.

[160] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, and Kurt Rothermel. "Distributed Spectral Cluster Management: A Method for Building Dynamic Publish/Subscribe Systems". In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. DEBS '12. Berlin, Germany: ACM, 2012, pp. 213–224. ISBN: 978-1-4503-1315-5. DOI: 10.1145/2335484.2335508. URL: http://doi.acm.org/10.1145/2335484.2335508.

[161] Muhammad Adnan Tariq, Boris Koldehofe, Gerald G. Koch, Imran Khan, and Kurt Rothermel. "Meeting Subscriber-defined QoS Constraints in Publish/Subscribe Systems". In: *Concurr. Comput. : Pract. Exper.* 23.17 (Dec. 2011), pp. 2140–2153. ISSN: 1532-0626. DOI: 10.1002/cpe.1751. URL: http://dx.doi.org/10.1002/cpe.1751.

[162] A. R. Thakar, A. S. Szalay, P. Z. Kunszt, and J. Gray. "The Sloan Digital Sky Survey Science Archive: Migrating a Multi-Terabyte Astronomical Archive from Object to Relational DBMS". In: *Comp. Sci. and Eng*. Vol. 5. 5. 2003, pp. 16 –29.

[163] Birjodh Tiwana, Mahesh Balakrishnan, Marcos K. Aguilera, Hitesh Ballani, and Z. Morley Mao. "Location, Location, Location!: Modeling Data Proximity in the Cloud". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 15:1–15:6. ISBN: 978-1-4503-0409-2. DOI: 10.1145/1868447.1868462. URL: http://doi.acm.org/10.1145/1868447.1868462.

[164] Birjodh Tiwana, Mahesh Balakrishnan, Marcos K. Aguilera, Hitesh Ballani, and Z. Morley Mao. "Location, location, location!: modeling data proximity in the cloud". In: *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. Hotnets-IX. Monterey, California: ACM, 2010, 15:1–15:6. ISBN: 978-1-4503-0409-2.

[165] Ankit Toshniwal, Siddarth Taneja, Amit Shukla, Karthik Ramasamy, Jignesh M. Patel, Sanjeev Kulkarni, Jason Jackson, Krishna Gade, Maosong Fu, Jake Donham, Nikunj Bhagat, Sailesh Mittal, and Dmitriy Ryaboy. "Storm@Twitter". In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. SIGMOD '14. Snowbird, Utah, USA: ACM, 2014, pp. 147–156. ISBN: 978-1-4503-2376-5. DOI: 10.1145/2588555.2595641. URL: http://doi.acm.org/10.1145/2588555.2595641.

[166] Durga Toshniwal and Ramesh C. Joshi. "Finding Similarity in Time Series Data by Method of Time Weighted Moments". In: *Proceedings of the 16th Australasian Database Conference - Volume 39*. ADC '05. Newcastle, Australia: Australian Computer Society, Inc., 2005, pp. 155–164. ISBN: 1-920-68221-X. URL: http://dl.acm.org/citation.cfm?id=1082222.1082239.

[167] Emalayan Vairavanathan, Samer Al-Kiswany, Lauro Beltrão Costa, Zhao Zhang, Daniel S. Katz, Michael Wilde, and Matei Ripeanu. "A Workflow-Aware Storage System: An Opportunity Study". In: *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*. CCGRID '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 326–334. ISBN: 978-0-7695-4691-9. DOI: 10.1109/CCGrid.2012.109.

[168] Vytautas Valancius, Cristian Lumezanu, Nick Feamster, Ramesh Johari, and Vijay V. Vazirani. "How Many Tiers?: Pricing in the Internet Transit Market". In: *SIGCOMM Comput. Commun. Rev.* 41.4 (Aug. 2011), pp. 194–205. ISSN: 0146-4833. DOI: 10.1145/2043164.2018459.

[169] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. "A Break in the Clouds: Towards a Cloud Definition". In: *SIGCOMM Comput. Commun. Rev.* 39.1 (Dec. 2008), pp. 50–55. ISSN: 0146-4833. DOI: 10.1145/1496091.1496100. URL: http://doi.acm.org/10.1145/1496091.1496100.

[170] Vinod Kumar Vavilapalli, Arun C. Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, Bikas Saha, Carlo Curino, Owen O'Malley, Sanjay Radia, Benjamin Reed, and Eric Baldeschwieler. "Apache Hadoop YARN: Yet Another Resource Negotiator". In: *Proceedings of the 4th Annual Symposium on Cloud Computing*. SOCC '13. Santa Clara, California: ACM, 2013, 5:1–5:16. ISBN: 978-1-4503-2428-1. DOI: 10.1145/2523616.2523633. URL: http://doi.acm.org/10.1145/2523616.2523633.

[171] *VB: Cloud layers*. http://venturebeat.com/2011/11/14/cloud-iaas-paas-saas/.

[172] Uri Verner, Assaf Schuster, Mark Silberstein, and Avi Mendelson. "Scheduling Processing of Real-time Data Streams on Heterogeneous multi-GPU Systems". In: *Proceedings of the 5th Annual International Systems and Storage Conference*. SYSTOR '12. Haifa, Israel: ACM, 2012, 8:1–8:12. ISBN: 978-1-4503-1448-0. DOI: 10.1145/2367589.2367596. URL: http://doi.acm.org/10.1145/2367589.2367596.

[173] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. "Characterizing Cloud Computing Hardware Reliability". In: *Proceedings of the 1st ACM Symposium on Cloud Computing*. SoCC '10. Indianapolis, Indiana, USA: ACM, 2010, pp. 193–204. ISBN: 978-1-4503-0036-0. DOI: 10.1145/1807128.1807161. URL: http://doi.acm.org/10.1145/1807128.1807161.

[174] Jens-Sönke Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge, and Bruce Berriman. "Experiences Using Cloud Computing for a Scientific Workflow Application". In: *Proceedings of the 2Nd International Workshop on Scientific Cloud Computing*. ScienceCloud '11. San Jose, California, USA: ACM, 2011, pp. 15–24. ISBN: 978-1-4503-0699-7. DOI: 10.1145/1996109.1996114. URL: http://doi.acm.org/10.1145/1996109.1996114.

[175] Jianwu Wang, Daniel Crawl, and Ilkay Altintas. "Kepler + Hadoop: A General Architecture Facilitating Data-intensive Applications in Scientific Workflow Systems". In: *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*. WORKS '09. Portland, Oregon: ACM, 2009, 12:1–12:8. ISBN: 978-1-60558-717-2. DOI: 10.1145/1645164.1645176. URL: http://doi.acm.org/10.1145/1645164.1645176.

[176] Zidong Wang, Xiaohui Liu, Yurong Liu, Jinling Liang, and Veronica Vinciotti. "An Extended Kalman Filtering Approach to Modeling Nonlinear Dynamic Gene Regulatory Networks via Short Gene Expression Time Series". In: *IEEE/ACM Trans. Comput. Biol. Bioinformatics* 6.3 (July 2009), pp. 410–419. ISSN: 1545-5963. DOI: 10.1109/TCBB.2009.5. URL: http://dx.doi.org/10.1109/TCBB.2009.5.

[177] Tom White. "Scalable Multi Swarm-Based Algorithms with Lagrangian Relaxation for Constrained Problems". In: *Hadoop: The definitive guide*. O'Reilly Media, 2009. URL: http://oreilly.com/catalog/9780596521981.

[178] Gary Wills, Victor Chang, and Robert John Walters. "Business Integration As a Service". In: *Int. J. Cloud Appl. Comput.* 2.1 (Jan. 2012), pp. 16–40. ISSN: 2156-1834. DOI: 10.4018/ijcac.2012010102. URL: http://dx.doi.org/10.4018/ijcac.2012010102.

[179] *WorldWide Telescope*. http://www.worldwidetelescope.org.

[180] *X Prize for Genomics*. http://genomics.xprize.org.

[181] Esma Yildirim and Tevfik Kosar. "Network-aware end-to-end data throughput optimization". In: *Proceedings of the first international workshop on Network-aware data management*. NDM '11. Seattle, Washington, USA: ACM, 2011, pp. 21–30. ISBN: 978-1-4503-1132-8. DOI: 10.1145/2110217.2110221. URL: http://doi.acm.org/10.1145/2110217.2110221.

[182] Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. "Discretized Streams: Fault-tolerant Streaming Computation at Scale". In: *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. SOSP '13. Farminton, Pennsylvania: ACM, 2013, pp. 423–438. ISBN: 978-1-4503-2388-8. DOI: 10.1145/2517349.2522737. URL: http://doi.acm.org/10.1145/2517349.2522737.

[183] Shams Zawoad, Amit Kumar Dutta, and Ragib Hasan. "SecLaaS: Secure Logging-as-a-service for Cloud Forensics". In: *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. ASIA CCS '13. Hangzhou, China: ACM, 2013, pp. 219–230. ISBN: 978-1-4503-1767-2. DOI: 10.1145/2484313.2484342. URL: http://doi.acm.org/10.1145/2484313.2484342.

[184] Xiaoxue Zhang and Feng Xu. "Survey of Research on Big Data Storage". In: *Proceedings of the 2013 12th International Symposium on Distributed Computing and Applications to Business, Engineering & Science*. DCABES '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 76–80. ISBN: 978-0-7695-5060-2. DOI: 10.1109/DCABES.2013.21. URL: http://dx.doi.org/10.1109/DCABES.2013.21.

[185] Yanfeng Zhang, Qixin Gao, Lixin Gao, and Cuirong Wang. "iMapReduce: A Distributed Computing Framework for Iterative Computation". In: *J. Grid Comput.* 10.1 (Mar. 2012), pp. 47–68. ISSN: 1570-7873. DOI: 10.1007/s10723-012-9204-9. URL: http://dx.doi.org/10.1007/s10723-012-9204-9.

Part V

Appendix

Chapter 15

Resumé

The easily-accessible computation power offered by cloud infrastructures coupled with the Data Deluge are expanding the scale and speed at which data analysis is performed. In their quest for finding the Value in the 5 Vs of Big Data, applications process larger data sets, within and across the globally distributed cloud data centers. Thus, to leverage the full computation power of the clouds, global data processing across multiple sites has to be fully supported. In this work we focus on the performance aspects of managing data, a key milestone for enabling such large-scale processing. To this purpose we proposed Tomus-Blobs, a concurrency-optimized PaaS-level cloud storage system which federates the virtual disks, leveraging data locality for cloud MapReduce or workflow processing. Encourage by the results, which showed a throughput increase of more than 2 times over the cloud storage, we have built a MapReduce framework that uses TomusBlobs as a data management back-end. To fully address the scaling needs of large Big Data analysis, we extended this framework to a multi-site hierarchical MapReduce engine, which enables efficient processing using resources from multiple cloud sites. The expensive inter-site data exchanges are minimized using the Map-IterativeReduce technique that we devised for reducing the number of output results of MapReduce processing to a single one. We illustrate the benefits of our approach with a concrete Big Data application, called A-Brain, from the area of joint genetic and neuroimaging data analysis. Our solutions permitted to run this analysis at large-scale on 1000 cores running for 2 weeks in 4 deployments across Azure data centers, one of the largest scientific experimental setup on Azure up to date. The analysis discovered for the first time how the functional signal in subcortical brain regions can be significantly fit with genome-wide genotypes. In addition to the biological result, we showed that our approach is fault tolerant and is able to provide reliable performance at large-scale despite distributing the computation across sites. Additionally, in the process of executing such Big Data analysis, we had the opportunity to gain useful experience and to learn several interesting lessons. The most important lesson was that enabling high-performance geographically data management and transfers in such scenarios becomes critical. In fact, geographically distributed processing becomes a reality both in the scientific world (e.g., genome mapping, high-energy physics simulations, large observatories sensors network) and in the commercial large IT web-services (e.g., search engines, office tools). To address these issues, we focused on understanding the options and strategies that can be set in place for managing application data, both statically and in real-time, across cloud data centers.First, we have investigated an alternative to collocating the storage in the federated disks of the compute nodes, in order to further increase the reliability while remaining non-intrusive. To this end, we propose a schema for selecting nodes to be dedicated for data management, by deducting in user space the topology of the cloud nodes. Our approach shows that it is possible to improves the overall performance of real-life scientific application with up to 45 %. Building on this uniform data management systems we device an environmentally-aware data handling approach for exploiting the network parallelism of clouds via multi-route transfers, while offering predictable cost and time performances. In terms of efficiency, it provides the applications with the possibility to set a trade-off between money and time and to optimize the transfer strategy accordingly, being able to reduce the monetary costs and transfer time by up to 3 times. Next, we have leveraged these strategies for the applications that require streaming the data to cloud compute nodes. For this we have proposed JetStream, a high-performance batch-based streaming middleware, that dynamically self-adapts to the streaming conditions by modeling and monitoring a set of context parameters. The results showed that by adaptively selecting the right configuration, JetStream is able to improve

the overall application streaming performance by as much as three times. Finally, considering all these approaches, we advocate for a Transfer as a Service (TaaS) paradigm that supports large-scale data dissemination across geographically distributed sites. We complement this with a dynamic cost model schema, which enables the cloud providers to regulate and encourage data exchanges via a data transfer market. Our work demonstrates that it is possible to adaptively exploit in user space the cloud resources in order to shift the providers advertised performances towards high-performance data management for the Big Data applications running across data centers.