Efficient support for data-intensive scientific workflows on geo-distributed clouds

Luis Eduardo Pineda Morales

Thèse

UNIVERSITE
BRETAGNE
LOIRE

THESE INSA Rennes
sous le sceau de l'Université Bretagne Loire
pour obtenir le titre de
DOCTEUR DE L'INSA RENNES
Spécialité : Informatique

présentée par

Luis Eduardo PINEDA MORALES

ECOLE DOCTORALE : *MATISSE*
LABORATOIRE : *IRISA*

Efficient Support for Data-Intensive Scientific Workflows on Geo-Distributed Clouds

Thèse soutenue le 24.05.2017
devant le jury composé de :

Pierre SENS
Professeur des universités, Université de Paris-VI / Président du jury
Frédéric DESPREZ
Directeur de recherche, Inria Grenoble – Rhône-Alpes / Rapporteur
Michael SCHÖTTNER
Professeur des universités, Universität Düsseldorf / Rapporteur
Esther PACITTI
Professeur des universités, Université Montpellier-II / Examinateur
Adrien LÈBRE
Chercheur, Inria Rennes – Bretagne Atlantique / Examinateur
Alexandru COSTAN
Maître de conférences, INSA de Rennes / Co-encadrant de thèse
Gabriel ANTONIU
Directeur de recherche, Inria Rennes – Bretagne Atlantique / Directeur de thèse

Efficient Support for Data-Intensive Scientific Workflows on Geo-Distributed Clouds

Luis Eduardo PINEDA MORALES

En partenariat avec

A Lucía, que no dejó que me rindiera.

Acknowledgements

One page is not enough to thank all the people who, in one way or another, have helped, taught, and motivated me along this journey. My deepest gratitude to them all.

To my amazing supervisors Gabriel Antoniu and Alexandru Costan, words fall short to thank you for your guidance, support and patience during these 3+ years. Gabriel: ¡Gracias! For your trust and support went often beyond the academic. Alex: it has been my honor to be your first PhD student, I hope I met the expectations.

To my family, Lucía, for standing by my side every time, everywhere; you are the driving force in my life, I love you. To my family in Mexico, who still blindly believes in me: my parents, my siblings, and my nephews; I love you too.

To Frédéric Desprez and Michael Schöttner for their time devoted to review this thesis and for their insightful comments. To them, Esther Pacitti, Adrien Lèbre and Pierre Sens for accepting to be part of my defense jury.

To the members of the Z-CloudFlow project: to Ji Liu for many days of partnership. To Patrick Valduriez and Esther (again) for hosting me in their lab in Montpellier. To Marta Mattoso for her feedback. To Radu Tudoran, who also guided my first steps into the PhD. To Laurent Massoulié and Pierre-Louis Xech from the Microsoft Research - Inria Joint Centre, for their support at different stages.

To Kate Keahey for welcoming me into the Nimbus team at ANL. To Balaji Subramaniam and Pierre Riteau for their mentorship, and to the CIGI team at UIUC for sharing their knowledge with us.

To Luc Bougé and Shadi Ibrahim for their always timely and wise advice. To my office mates Nat' and Yacine for the fun and the help, especially during the last stage of the thesis. To my honorary office mate, Lokman, for those afternoons filled with discussion and learning. To Orçun, with whom I probably spent more time than I should have. To Dat, with whom I should have spent more. To my fellow KerDatiens: Matthieu, Anirvan, Pierre, Ovidiu, Amelie, Hadi, and Paul; for their encouragement and good vibes, I cherish our time together. To Aurélie and Hélène, for their support with administrative issues.

To Nico, Belén, Antoine, Lolis and all my Breizh-Mex friends, who helped us build a new home away from home. To my Erasmus Mundus friends, who keep inspiring me. And to my friends in Mexico, who linger even if I always put them at the end of the acknowledgements.

Thank you!

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Contributions . . . 3
1.3 Publications . . . 4
1.4 Organization of the Manuscript . . . 5

Part I — **Context** **7**

2 Background: Data-Intensive Scientific Workflows on Clouds **9**
2.1 Large-Scale Data-Intensive Applications . . . 9
2.1.1 Scientific Applications . . . 10
2.1.2 Commercial Applications . . . 11
2.1.3 Challenges of Data-Intensive Applications . . . 11
2.2 Cloud Computing . . . 14
2.2.1 Types of Cloud Services . . . 14
2.2.2 Cloud Deployment Models . . . 15
2.2.3 Single- and Multisite Clouds . . . 17
2.2.4 Data Processing on the Cloud: MapReduce . . . 19
2.3 Scientific Workflows on Clouds . . . 21
2.3.1 Overview: The Scientific Workflow Model . . . 22
2.3.2 Scientific Workflows on Clouds: State of the Art . . . 26
2.4 Challenges Addressed . . . 29
2.4.1 Metadata Management . . . 29
2.4.2 Elastic Usage of Resources . . . 30

Part II — **Contributions** **31**

3 Metadata Management for Geo-Distributed Clouds **33**
3.1 Problem: Huge Latency for Remote Metadata Access . . . 33
3.2 Solution: Adaptive Decentralized Metadata Handling . . . 35
3.3 Design Principles . . . 35
3.4 Strategies for Multisite Metadata Management . . . 38
3.4.1 Centralized Metadata (Baseline) . . . 38

3.4.2 Replicated Metadata (On Each Site) . . . 39
3.4.3 Decentralized, Non-Replicated Metadata . . . 40
3.4.4 Decentralized Metadata with Local Replication . . . 40
3.5 Discussion . . . 41
3.5.1 Matching Strategies to Workflow Patterns . . . 41
3.5.2 Distributing Workflows Across Datacenters . . . 42
3.6 Conclusion . . . 43

4 One Step Further: Managing Hot Metadata 45
4.1 What is "Hot" Metadata? . . . 46
4.2 Design Principles . . . 46
4.3 Architecture . . . 48
4.3.1 Two-level Multisite Architecture . . . 49
4.3.2 Filter for Hot and Cold Metadata . . . 49
4.3.3 Decentralized Hot Metadata Management Strategies . . . 49
4.3.4 Multisite Hot Metadata Protocols . . . 50
4.4 Discussion . . . 51
4.4.1 Challenges for Hot Metadata Management . . . 51
4.4.2 Towards Dynamic Hot Metadata . . . 51
4.5 Conclusion . . . 52

5 Evaluating Elasticity Factors in Workflow Applications 53
5.1 Dynamic Data Streams: Motivations and Challenges . . . 54
5.2 Towards Smart Elastic Provisioning on Clouds . . . 55
5.2.1 Evaluating Scalability Factors for Smart Appliances . . . 55
5.2.2 Designing a Smart Elastic Provisioner . . . 56
5.3 Use Case: Spatial Data for Urban Dynamics . . . 58
5.4 Discussion: Metadata Role in Data Location-Aware Elasticity . . . 59
5.5 Conclusion . . . 59

Part III — **Implementation and Results 61**

6 Implementation Details 63
6.1 Distributed Metadata Registry . . . 63
6.2 DMM-Chiron . . . 65
6.2.1 Baseline: Multisite Chiron . . . 66
6.2.2 Metadata Manager . . . 67
6.2.3 Job Manager . . . 67
6.3 Conclusion . . . 68

7 Evaluation 69
7.1 Experimental Setup . . . 69
7.1.1 Infrastructure . . . 69
7.1.2 Use Cases . . . 70
7.2 Decentralized Metadata Management on the Cloud . . . 72
7.2.1 Impact of Metadata Decentralization on Makespan . . . 73

7.2.2 Scalability and Concurrency Sensitivity . . . 75
7.2.3 Support for Real-Life Workflows . . . 76
7.2.4 Discussion . . . 77
7.3 Separate Handling of Hot and Cold Metadata . . . 78
7.3.1 Hot Metadata for Different Workflow Structures . . . 78
7.3.2 Zoom on Multi-Task Jobs . . . 80
7.3.3 Discussion: Limitations to be Addressed . . . 81
7.4 Identifying Thresholds for Elastic Scaling . . . 83
7.4.1 Exploring Tradeoffs Related to Data Block Size . . . 83
7.4.2 Impact of the Data Replication Factor . . . 85
7.4.3 Parameters Not Modelled . . . 87
7.4.4 Discussion: *Smart* Policies for *Smart* Provisioning . . . 88
7.5 Conclusion . . . 88

Part IV — **Conclusions and Perspectives** **91**

8 Conclusions **93**
8.1 Achievements . . . 94
8.1.1 Enabling Multisite Metadata Management for Workflows . . . 94
8.1.2 Managing Hot Metadata Across Datacenters . . . 94
8.1.3 Studying Elasticity Parameters to Steer Smart Scaling . . . 95
8.2 Future Work and Perspectives . . . 95
8.2.1 Improving Multisite Metadata Management . . . 95
8.2.2 Enabling Smart Elastic Provisioning . . . 96
8.2.3 Workflow Management: From Batch to Stream Processing . . . 96

Bibliography **97**

Part V — **Appendix** **107**

Résumé en Français **109**

Chapter 1

Introduction

Contents

1.1 Context 1
1.2 Contributions 3
1.3 Publications 4
1.4 Organization of the Manuscript 5

1.1 Context

We live in an era where data volumes are so huge and grow so fast that, in their popular yearly report, IDC Research[1] calls them the *Digital Universe* [62], stating that by 2020 there will be "*nearly as many digital bits as there are stars in the [physical] universe*". The same study estimates that the digital universe will reach 44 zettabytes by that year, while Facebook alone reported a daily production of 4 PB in 2014 [142]. Moreover, such data come in the most diverse shapes and from the most geo-distributed sources ever. The data explosion calls for computing resources with correspondingly expanding storage and processing power, and for applications capable of highly-scalable, distributed computation in order to leverage those resources.

Countless scientific and business applications deal already with (very) large fractions of that digital universe. With some datasets reaching the order of petabytes, these applications require high-end environments to run; traditionally, they are hosted either in *supercomputers* or *clouds*. Modern supercomputers reach performances in the order of tens of petaflops [133]; however, the access to one of these computing colossus is somewhat exclusive since purchasing and maintaining a supercomputer is out of the budget for an average company/institution. Furthermore, even when a supercomputer is available, its service is restricted and prioritized to limited groups of users.

[1] International Data Corporation Research.

Clouds, on the other hand, have emerged as a cost-effective alternative for intensive computing. In most cases, they consist of several *datacenters* (or *sites*) scattered in different geographic areas. They offer the possibility to lease a large, dynamic amount of federated resources for just the necessary period of time, and at relatively affordable prices; while keeping hardware management transparent to the user. Computing clouds have enabled both business and academia to expand their potential to carry out large-scale computations without impacting their finances.

These large-scale applications usually consist of lots of processing threads running simultaneously on multiple machines. A common approach to implement such applications is using the well-known MapReduce programming model [39], which allows to perform parallel computations over partitions of a large dataset and then merge the obtained partial results. However effective this model is for numerous problems, it does not match many other cases where, for instance, the application flow includes heterogeneous dependency patterns, or data are produced/stored in several geographic locations.

In order to overcome the limitations of MapReduce, an increasingly adopted practice is to use *workflows*. A workflow is a more abstract model to formally specify an application as a directed graph, where vertices denote computational tasks and edges the data dependencies between such tasks. With this loosely-coupled model we are able to clear the expressiveness barrier for multi-step computational tasks; yet, a critical issue remains: large-scale applications are not only complex to model, but also resource-greedy. Many of these applications process petascale volumes, the ALICE experiment at the CERN, for example, requires the capacity to store up to 1.25 GB every second [26], let alone the capacity to analyze those data. Thus, in the practice, for several scenarios it turns impossible to run an application in a single location; either because the data/computation are just too large/extensive to fit in the available resources, or perhaps because all the data needed are rather dispersed over the globe. As a result, a new factor comes into play: *multisite processing*.

To date, some cloud providers offer means to communicate between datacenters, e.g. data buses [79], or dedicated networks [5], partially bypassing the overhead of manually setting inter-site network configurations. However, a more concerning drawback in multisite processing is that inter-site network latencies are orders of magnitude higher than those within a single datacenter. As workflows require large amounts of data to be transferred from a task to the next one, what in a single site could take no time by using shared file systems, might translate into severe network congestion and delays in a multisite setting, especially when IP traffic is predicted to increase threefold over the next five years [34].

There is no doubt that clouds have facilitated multisite deployments to an extent; still, several challenges arise to cope with the foregoing handicaps and enable workflow execution across datacenters. Specifically:

1. How to minimize the communication overhead between datacenters?
2. How to realize workload balancing between datacenters to avoid bottlenecks?
3. How to optimize cloud resource usage to reduce costs?
4. What strategies to use for big data storage and transfer?
5. How to group tasks and datasets together to minimize transfers?

While all of these challenges steer and motivate the present thesis, our contributions, introduced in the next section, target in particular items 1, 2 and 3. Challenges 4 and 5 have been addressed in previous theses [134, 71] within the same project as ours: "Z-CloudFlow: Data Workflows in the Cloud" of the Microsoft Research – Inria Joint Centre.

1.2 Contributions

Our research contributions are summarized below. The first two correspond to work carried out within the Z-CloudFlow project. The last one is related to an internship at Argonne National Laboratory, USA, in the context of the Joint Laboratory for Extreme Scale Computing (JLESC) and the Data@Exascale associate team.

Improving Cloud Workflow's Performance Through Adaptive Distributed Metadata Management

In a typical run of a large-scale application, the dataset gets to be partitioned in many chunks for parallel processing. In the case of workflows, the input data often consist of a significant amount of rather small files, and grows as the workflow progresses and intermediate data are generated. In either case, huge loads of metadata are required to keep track of each data unit (files, chunks, records) and of the whole execution state. Such metadata overload can easily saturate state-of-the-art file systems, which are mostly designed for single-site premises and metadata are managed in a centralized server (if any). The scenario gets more complex if the application is deployed in a multisite cloud: each metadata update has to be registered to a potentially remote centralized server through a high-latency network. Even if these updates are done in batches, it is unquestionable that the performance of the application will significantly drop.

In order to reduce that impact, we explored design strategies that leverage workflow semantics in a 2-level metadata partitioning hierarchy that combines distribution and replication. We implemented such strategies in a multisite cloud to support workflow execution and validated our approach across four datacenters using synthetic benchmarks and real-life applications. We were able to obtain as much as 28 % gain in execution time for a parallel, geo-distributed real-world application and up to 50 % for a metadata-intensive synthetic benchmark, compared to a baseline centralized configuration. This work addresses challenge 2, and has been published in [Cluster2015].

Enabling Efficient Propagation of Frequently Accessed Workflow Metadata

With the massive volumes of data handled nowadays by large-scale computing systems, an analogous increase in the load of metadata has brought their designers' attention to their metadata handlers. The number of metadata operations grows exponentially with the number of data items; as a consequence, poor or non-existing metadata management strategies generate bottlenecks that might impact the system's performance. Dedicated metadata management is the key to success in some file systems [132]; however, their solutions apply mostly to single-site, HPC infrastructures to date.

As we are concerned with multisite cloud workflow applications, this contribution extends the aforementioned hybrid decentralized/distributed metadata handling model, tackling challenge 1. We analyzed workflow metadata by their frequency of access and denote *hot metadata* to the most frequently required (conversely, *cold metadata*). We developed an approach that enables timely propagation of hot metadata while delaying cold metadata operations. This action reduces network congestion by limiting the number of operations sent through high latency networks, thereby improving the overall workflow execution time.

Moreover, we coupled our design with a workflow execution engine to validate and tune its applicability to different real-life scientific scenarios. Our results revealed a consistent improvement of over 20 % for highly parallel workflow jobs, which are a constant in large-scale processing. A paper describing this contribution was accepted to [BigData2016].

Evaluating Elasticity Factors for Smart Appliances in Private Clouds

A key characteristic of clouds is *elasticity*, i.e. the ability to provision and de-provision computing resources (usually virtual machines) in response to the current workload. In general, the new machines are instantiated from images pre-configured with the required software and settings, called *appliances*. Workflows can significantly benefit from elasticity, as the number of allocated machines could be dynamically adjusted according to the resource usage of each task, avoiding charges for idle time. In order to achieve this dynamic response, we must first identify the parameters that burst/decrease resource consumption in an application. Spotting and analyzing such parameters will allow to model their behavior along the execution, with the ultimate goal of predicting the right timing for elastic scaling.

In this contribution we leveraged a social analysis workflow application to pinpoint elasticity factors, which relate to challenge 3. We provided and tested private cloud resources for enabling spatial (geo-tagged) data synthesis. Specifically, the application evaluates Twitter data to estimate individuals' home and work relocation, and correlate them with unemployment rates. We realized an analysis of the different stages of the application in terms of data loading and replication, execution time and parallelism. Then, we presented details into tuning these configuration parameters to extract the best performance. We identified under-utilized resources, tradeoffs for data partitioning and replication, and points of resource saturation for up to 42 days of data (~80 GB); all of which are factors that elicit elastic scaling. This work has been presented in [SC2015].

1.3 Publications

Papers in International Conferences

[BigData2016] Luis Pineda-Morales, Ji Liu, Alexandru Costan, Esther Pacitti, Gabriel Antoniu, Patrick Valduriez and Marta Mattoso. *Managing Hot Metadata for Scientific Workflows on Multisite Clouds*. In IEEE International Conference on Big Data, Dec 2016, Washington, United States.

[Cluster2015] Luis Pineda-Morales, Alexandru Costan and Gabriel Antoniu. *Towards Multisite Metadata Management for Geographically Distributed Cloud Workflows*. In IEEE International Conference on Cluster Computing, Sep 2015, Chicago, United States.

Posters in International Conferences

[SC2015] Luis Pineda-Morales, Balaji Subramaniam, Kate Keahey, Gabriel Antoniu, Alexandru Costan, Shaowen Wang, Anand Padmanabhan and Aiman Soliman. *Scaling Smart Appliances for Spatial Data Synthesis*. In ACM/IEEE International Conference in Supercomputing, Nov 2015, Austin, United States.

[Cluster2014] Luis Pineda-Morales, Alexandru Costan and Gabriel Antoniu. *Big Data Management for Scientific Workflows on Multi-Site Clouds*. In PhD Forum, IEEE International Conference on Cluster Computing, Sep 2014, Madrid, Spain.

1.4 Organization of the Manuscript

The rest of the manuscript is organized as follows.

The first part provides the necessary context to our research in data-intensive workflows on clouds. In Chapter 2 we first characterize different sorts of large-scale applications. Then, we focus on the cloud computing paradigm and principles and explain the types and levels of services available. Finally, we give an overview of workflows and their features, together with a survey on the state-of-the-art workflow management systems, highlighting the main challenges addressed by this work.

The second part consists of the three contributions of the thesis. In Chapter 3 we present several alternative design strategies to efficiently support the execution of existing workflow engines across multisite clouds, by reducing the cost of metadata operations. In Chapter 4 we take one step further and explain how selective handling of metadata, classified by frequency of access, improves workflows performance in a multisite environment. Finally, in Chapter 5 we look into a different approach to optimize cloud workflow execution by evaluating execution parameters to predict elastic scaling.

The third part presents the implementation and evaluation details of our contributions. In Chapter 6 we first present a distributed metadata registry that implements the different hybrid strategies for metadata storage. Then, we add a filter that discriminates metadata by access frequency and couple it with a workflow management system. Afterwards, in Chapter 7 we provide an extensive evaluation of our system, comparing it to state-of-the-art baselines, covering both synthetic and real-life workflows in public and private multisite clouds.

The fourth part contains Chapter 8, which concludes this thesis and provides a perspective of future research directions in the field.

Part I

Context

Chapter 2

Background: Data-Intensive Scientific Workflows on Clouds

Contents

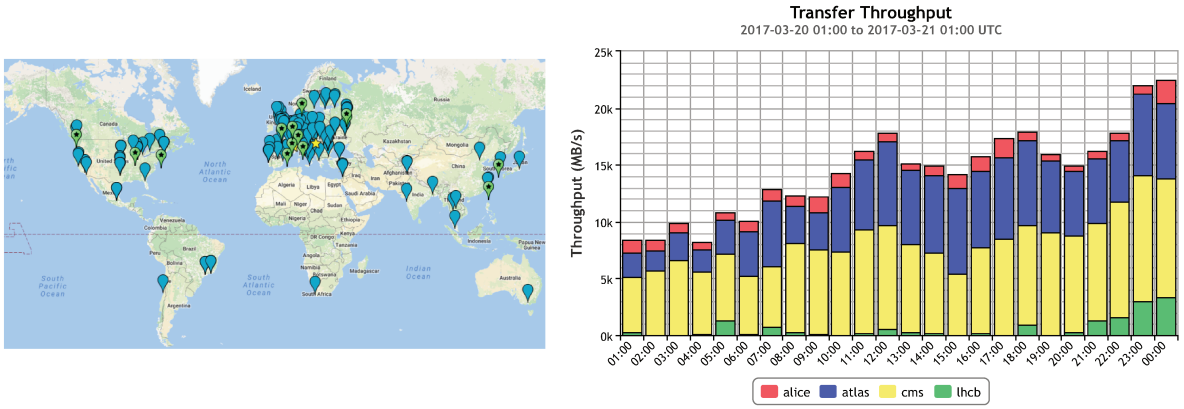
2.1 Large-Scale Data-Intensive Applications 9
2.2 Cloud Computing 14
2.3 Scientific Workflows on Clouds 21
2.4 Challenges Addressed 29

BIG DATA is the buzzword that is driving the way we do business and science. If we randomly pick a knowledge domain, we will find data being generated in massive amounts and at an increasing rate, in industry and research alike. The large-scale applications that handle these volumes of data require appropriate programming models that allow for heavily-parallel computing and process automation. Moreover, robust infrastructures are needed to uphold the implicit computing and storage demands.

The present thesis is motivated by that need for efficient techniques/resources to support data-intensive applications, as we elaborate in this chapter. In order to set a context for our research agenda, we first provide some examples of *large-scale applications* in different domains and discuss the inherent challenges of big data processing. Then, we introduce *cloud computing*, a suitable infrastructure model for big data processing. Finally, we direct the focus of this manuscript towards a specific category of loosely-coupled, formally-defined applications: *workflows*.

2.1 Large-Scale Data-Intensive Applications

The concept of *large-scale* computing is certainly relative to a time window. The computing capacity supplied by the five IBM System/360 mainframes that calculated the trajectory of



(a) Distribution of the 170 computing centers.

(b) Transfer throughput in a 24-hour window.

Figure 2.1 – The Worldwide LHC Computing Grid, hosted in 42 countries.[1]

the Apollo 11 [61], was far much smaller than that of any mobile device available today. Moore's Law, that ruled the pace of transistors shrinkage for 50 years, is breaking down to give way to a new generation of more capable computers that cope with today's processing needs [128]. Even the term "*petascale*", that evoked the most demanding computing tasks and the most powerful machines, had to evolve to *exascale*, and to *extreme scale* computing [117] to reflect the vast data proliferation and the capacity required by the applications that process them.

To give an idea of the dimension of current data-intensive applications, we present a selection of scientific and commercial scenarios that require cutting-edge, large-scale technological solutions.

2.1.1 Scientific Applications

LHC. The Large Hadron Collider [27] is the world's largest particle accelerator, it has fostered physics research and led to groundbreaking discoveries like the Higgs boson in 2013. Lying at the Swiss-French border in the CERN, the LHC is likely the flagship of data-intensive applications. Even after filtering out 99 % of their collected data, the LHC gathered ~50 petabytes of data in 2016. Because CERN does not have the computing resources to process that amount of data, they rely on a global computing infrastructure, the Worldwide LHC Computing Grid (WLCG) [28]. The WLCG spans over 42 countries and 170 computing centers to aggregate resources for data storage and analysis (Figure 2.1.a). The transfer throughput from the various experiments averages 10 GB/s at mostly any given time (Figure 2.1.b). ATLAS and CMS, for example, are collision detectors recording *massive flows of data* for experimental physics. ATLAS trigger [29] selects 100 events out of 1 billion every second; at this rate, the *decision* of keeping an event is made in less than 2 μs. The CMS tracker [30] records paths of particles moving almost at the speed of light, with an *accuracy* of 10 μm.

[1] Images are property of CERN — European Organization for Nuclear Research.

Genomics. DNA sequencing is a key process in biological research and its related fields. Notably, the ability to sequence human genome is leading to advances in diagnosis and treatment of diseases. Currently, one individual's genome represents approximately 100 GB. Since the first full sequences achieved at the beginning of the millennium, the progress in sequencers technology has made the cost of obtaining one genome drop from 1 million to 1 000 dollars [99]. In addition, top genomics institutes are now able to produce 2 000 genomes a day [114]. These numbers have triggered ambitious public-health initiatives, aiming to sequence up to 1 million genomes [53, 118] in favor of research for cancer and rare diseases. Such projects would reach the order of petabytes solely for genome storage. The field of genomics calls for computing infrastructures that can keep up with its *data explosion*.

2.1.2 Commercial Applications

Facebook is at the moment the largest social network, it serves an average of 1.23 billion daily users [47], who generate as many as 4 million *likes* per minute [23] and 350 million photo uploads per day [130]. Facebook uses a social graph store, TAO [22], to display everyone's personalized news feed, which contains thousands of individual objects. Friendships, likes, check ins, tags, and comments are stored as edges of an enormous graph. TAO accesses tens of petabytes of *small data*, performing billions of queries per second across eight globally distributed data centers [142]. With a ratio of 500 reads for every write, Facebook strongly relies on caching to reduce the overload in their storage systems [98]. The proper operation of Facebook is ensured through real-time analysis and performance monitoring done entirely in memory, using hundreds of servers of 144 GB RAM each [1].

Aeronautics is yet another industry in which data are massively produced and critical processes require real-time data analysis. The Airbus A350 is equipped with 6 000 sensors, generating 2.5 TB of data per day and this value is expected to triple in their new aircraft series [75]. The engines capture details all along their operation, in order to assess the impact of environmental conditions during the flight. Continuous monitoring of the aircraft components coupled with advanced data analytics drive *predictive maintenance* [37], i.e. timely replacement of faulty parts, which would otherwise take months. On a tightly related business, Heathrow airport, the most congested on Earth, is using big data processing to improve the connection times of 75 million passengers per year [38]. Big data is thereby helping aeronautics to reduce costs, improve safety and provide a better user experience.

As we can see from the above scenarios, big data brings about scientific and business progress. However, this positive impact comes at a cost: computer scientists face several challenges in their journey to design a new generation of systems for data-intensive computing. Systems that have to make immediate decisions, highly accurate measurements, or provide safety features, while dealing with overwhelming volumes of heterogeneous data. In the remainder of this section we discuss such challenges.

2.1.3 Challenges of Data-Intensive Applications

The 3 (or 5) Vs. Big data is defined by three key characteristics: volume, velocity and variety [148], often called the *3Vs*. These features represent also some of the main challenges

for large-scale data processing. Some authors add variability and veracity to complete a 5-element set. We make a focus on the core three.

Volume. In many fields, data recording devices (scanners, radars, sensors or applications) are evolving faster than the infrastructures to process such data. Measurements are more accurate and data production is cheaper; this is added to the apex of social media that has turned humans into an endless source of data (*crowdsourcing*). Consequently, much larger volumes of data are generated. Until recently, the main volume-related concern was *how* to store the data (what structure, what infrastructure). However, the growth in data volumes has gone beyond that challenge, to the issue of *when* to process them. Due to the lack of proper infrastructure, it is taking much longer to analyze the data than to obtain it. Genomics data can be sitting in a storage for over one year before being analyzed [114], which is clearly slowing down scientific discovery.

Velocity. One issue related to velocity was discussed above: data are produced so fast that the underlying infrastructure can not keep the pace to process it. Another concerning challenge is the speed at which a system has to react to the data production. A large number of applications are moving from static batch processing to *real-time* processing of sequences of events (*streams*) [14]. These scenarios require immediate response as the data are being generated, and range from "simple" video-on-demand services like Netflix or YouTube [116], to discovery-enabling devices like ATLAS, to vital on-flight analysis in aircrafts. All of them rely on highly-reactive systems that process events in milli- to microseconds to perform correctly.

Variety. Data are generated in the most diverse formats: from plain text to multimedia files, using generic exchange languages (like XML) or proprietary layouts. They can be stored in relational or NoSQL databases, or simply as binary objects. Using the Facebook example we can observe that this data variety can happen within a single scenario: from live streams to static media and *friends* databases; from 4-GB, two-hour videos [48], to tiny *likes*. Integrating and interoperating all these sorts of data representations might require complicated mechanisms of data conversion.

Dispersion. The data used by a single application might be produced and stored in many distant locations. In the case of Facebook, for example, data are stored in multiple data centers in US and Europe. They require specialized infrastructure and proprietary tools for performing high-speed queries on those data across several locations. Seismological sensors are dispersed over wide geographic areas reporting events in near-real time. More simply, an experiment might need combine data from different remote sources. Due to the high-latency networks used to transfer those data (usually through the public Internet), the capacity for processing and analysis is often hindered by the capacity to gather them.

Durability. Data durability is crucial at any scale; many systems ensure it through different techniques for data replication [144, 129]. These methods were efficient for years, but with very large datasets, replica control is a challenge. Moreover, data replication implies that the limit of the storage capacity is reached much faster, facing the users with a dilemma of whether to pay for additional storage or define eviction policies to discard the least relevant data.

I/O. Sometimes, big research centers actually have the resources to deal with massive data processed at high speeds. It is then, when neither storage capacity nor computing power are an issue, that I/O throughput can turn into a performance bottleneck. The most efficient machine, processing data kept in the largest storage system, will be always limited by the speed at which these data are read/written. Some mechanisms are tackling this challenge by using dedicated cores for I/O [44].

Security. It is often impossible for organizations to store their large volumes of data in their own premises. Hence, they are forced to lease storage space from a third party. Usually, the data transfers to the leased infrastructure is done through the public Internet, leaving the data vulnerable to attacks. This implies the overhead of encrypting/decrypting data for companies dealing with sensitive data like banking details, or health records used, for example, in genomics studies.

How to leverage infrastructures? Large-scale applications need specialized infrastructures to run. Depending on the data size and the available budget, an application can be deployed to small or mid-sized clusters of commodity machines. However, two infrastructures are preferred (or even required) for applications processing data in the order of hundreds of terabytes or more, and whose execution requires a large amount of computing power: supercomputers and clouds.

HPC. Supercomputers are processing titans. They consist of arrays of multi-core computing nodes, enhanced with performance-boosting features (e.g. GPUs) and interconnected by very high-speed networks. Supercomputers are conceived to execute millions of operations per second, the world's top supercomputers reach now the order of petaflops. As such, they can process huge amounts of data in reasonable periods of time. A wide range of large-scale scientific applications involving simulations, visualization and analytics are hosted in supercomputers: climate simulations for disaster prediction, biomolecular systems, or physics from sub-atomic particles to galaxies [18].

However, having this power requires significant upfront and periodic monetary investments: supercomputers themselves are fairly expensive ("low-cost" machines starting at half a million dollars [35]), the room to place them requires devices that ensure adequate conditions of humidity and temperature, the operation of the whole infrastructure generates elevated electricity bills, and the maintenance involves hiring the services of an expert. Moreover, if the data to process is distributed in different locations, the power of the supercomputer is limited by the speed of the networks that bring the data closer. Supercomputers are the norm for extreme scale computing; however, they do not fit every class of problems and, in particular, they do not fit every budget.

Cloud. Cloud computing has emerged as an affordable alternative to supercomputers. In a cloud, users can rent computing resources for limited periods of time. Clouds eliminate all the burden of setting up and maintaining a supercomputer, enabling their users to focus only on their business/research. Additionally, many cloud providers have premises in different locations, what allows to process data closer to where they are stored or generated.

In this thesis we adopt cloud computing as our base infrastructure model. This decision comes from the fact that our target applications handle data which are naturally dispersed, what makes them more suitable for clouds than for supercomputers. In the next section we make a thorough review of the concept of cloud computing.

The choice of an infrastructure depends on several factors: the organization's budget, for sure, but also the nature of the problems to be addressed, the business goals and the expected evolution of the project/company. Expanding organizations continuously evaluate the tradeoff between using the services of a cloud provider or investing in privately owned infrastructures [67].

2.2 Cloud Computing

Cloud computing is a model for *on-demand access* to a pool of computing, storage and network resources through the Internet. These resources can be elastically provisioned and removed as needed and users are billed only for the capacity used on a *pay-as-you-go* basis [78]. The cloud *provider* administers the resources and establishes service level agreements (SLAs) to guarantee the availability and quality of their service. The paradigm of cloud computing is revolutionizing the worlds of business and academia.

- Clouds have opened the possibility to *manipulate large, powerful and dynamic collections of resources*, eliminating upfront expenses like infrastructure purchases, and long-lasting expenses like electricity or maintenance bills.
- Cloud providers offer fully-managed platforms for data processing, storage and analysis, *allowing users to deploy applications faster and focus on their market goals* rather than on infrastructural design.

These conditions have enabled organizations to perform large scale data manipulation that previously was financially unviable and, as a consequence, accelerated business development and scientific discovery.

2.2.1 Types of Cloud Services

Depending on the degree of abstraction and control that they present to the user (Figure 2.2), cloud services are categorized in three levels of what is often called the *cloud computing stack*, as each builds on top of the previous one.

IaaS — Infrastructure as a Service. It is the bottom layer of service, users are given the capability to create and manage raw processing (virtual machines), storage or network components; but also the responsibility for any additional configuration required to operate them. The cloud provider maintains a catalog of infrastructure resources, such as operating system images, disks or dedicated networks, from which the resources can be pooled. While warranties exist in terms of availability and fault tolerance, there is no liability for the provider with respect to the interconnectivity of the components or their correct performance with a given application.

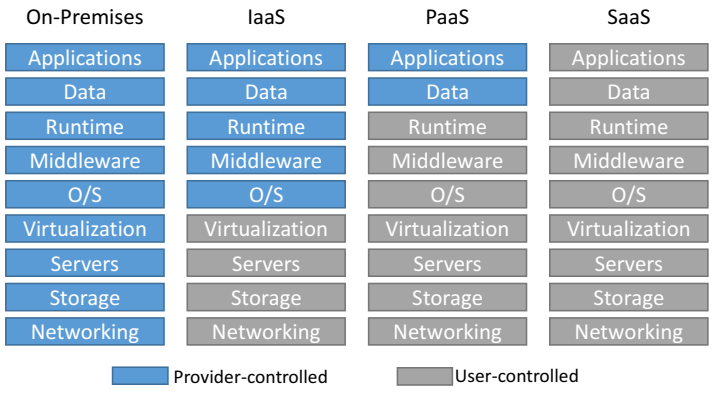


Figure 2.2 – Degree of user control in the different levels of the *cloud stack*.

PaaS — Platform as a Service. At this level, the users have access to a variety of computing platforms, in which they can create or execute their own applications. These platforms are generally managed by the provider, under certain security and availability agreements. These warranties are ensured through some encryption and/or replication techniques, but transparently for the users. An example of these services is Azure's HDInsight [80], which features tools for creating Hadoop clusters for the Apache Big Data processing stack (Pig, Hive, Spark, etc.).

SaaS — Software as a Service. It presents the highest level of abstraction for the users and requires minimum back-end configuration. Software applications (sometimes called *services*) are hosted on top of fully managed platforms and their underlying infrastructures. These applications are usually accessible through web browsers, web services, or custom interfaces like a smartphone app. One example of SaaS standing on top of the other layers is Microsoft Office 365, which uses Azure Cloud's PaaS (Azure Active Directory) for authentication [81], and IaaS (Azure Blobs) for storage [82], in a transparent way for their users.

2.2.2 Cloud Deployment Models

According to the accessibility of their resources, we distinguish three deployment models for clouds: public, private and hybrid. We present them hereunder and list representative examples or scenarios for each of them.

Public Clouds

Public clouds are multi-tenant environments open to the general public. They are owned and operated by third-party providers which are in charge of the maintenance of the infrastructure. Users normally require an account and a payment method as the resources are charged in a pay-as-you-go fashion. The allocation units vary from provider to provider and depending on the resource, virtual machines can be charged per hour or per minute, while storage is billed in GB or PB/month [83].

Three companies lead the market of public clouds: Amazon, Microsoft and Google. At the IaaS level, they all provide services for computing and storage, accessible through a

dashboard, APIs and command line. At the PaaS level, they offer fully managed frameworks to create, host and scale applications. In addition, each of them has particular features that make them global referents.

Amazon Web Services. Also known by its acronym, AWS, Amazon Web Services [6] is a suite of cloud computing services owned by Amazon.com, providing solutions at the IaaS and PaaS levels. Launched in 2006, it could be consider as a pioneer provider of public cloud resources. Amazon Elastic Compute Cloud (EC2) [7] is among its most popular services, it provides virtual machines for leasing, with user-controlled geolocation and redundancy. Amazon Simple Storage Service (S3) [8], is a scalable object storage accessible through web services interfaces (REST, SOAP) and the BitTorrent protocol. Both EC2 and S3 are part of the IaaS layer. On the PaaS side, Amazon's Elastic MapReduce (EMR) [9] was also the first public fully-managed Hadoop platform.

Microsoft Azure. Initially released as Windows Azure and later renamed for strategic purposes, Azure [84] offers the three levels of service with global presence. The bottom IaaS provides a large OS image repository and a range of virtual machine sizes for different computation purposes. Azure Storage [85] accounts for diverse solutions for every need: raw disk storage for I/O intensive applications, binary large objects (BLOBs) for unstructured data, distributed file systems, and NoSQL tables, to name a few. Interesting PaaS include the aforementioned HDInsight [80] for Hadoop-based applications and Azure App Service [86], a common platform to create different types of applications (API, web, mobile and business workflows). Finally, SaaS is delivered through many of Microsoft's popular products, such as Office 365, Xbox and Skype which are (partially) powered by Azure.

Google Cloud Platform. Google was the last of the three top providers to release their public cloud [55]. One distinctive feature of their IaaS Compute Engine is the possibility to create low-cost, preemptible virtual machines for fault-tolerant batch jobs [56]. Google had a breakthrough by making available a cloud version of their proprietary Bigtable [32], a distributed storage system for structured data used in many of their products, like Google Maps and Google Analytics. Another prominent PaaS is BigQuery [57], a scalable data warehouse for analytics which is becoming increasingly popular, probably due to Google's reputation in data analytics.

Private Clouds

The use of this cloud infrastructure is exclusive to a single tenant (company, organization), whether the resources are hosted within the tenant's premises or at a third-party provider. Private clouds customers trade the higher cost of dedicated infrastructure for specific features.

Security. The infrastructure can be restricted to other users through firewalls or custom authentication methods, or might be even physically isolated.

Availability. The tenant has control on the whole cloud without the need to book or compete for shared resources.

Customization. Sometimes, lower level custom configurations are not allowed in shared infrastructures. A private cloud allows to further personalize the resources.

Some literature consider *Community Clouds* as category on its own. Community clouds are private clouds shared by a group of users with common interests (e.g. business, research). The infrastructure is usually hosted on the premises of one of the members of the community and the costs are shared or subsidized. An example of said type of clouds is Chameleon [31], shortly described below.

Private clouds are sometimes managed with proprietary applications; however, a recurrent practice is to use public cloud managing software like OpenStack, which we briefly characterize next.

Chameleon. The Chameleon testbed [31] is a facility for scientific experimentation, it is deployed at two locations, one at the University of Chicago and another at the Texas Advanced Computing Center (TACC), interconnected by a 100 Gbps network. It accounts for ~650 multi-core nodes, and 5 PB of disk space. It offers two levels of infrastructure: reconfigurable bare-metal and OpenStack KVM virtual environment. As opposed to commercial clouds, Chameleon, conceived for scientific use, incorporates features and levels of control for advanced users, such as: customizable kernels, testbed versioning, isolated partitions, InfiniBand network or GPUs.

OpenStack. Although not a cloud provider itself, OpenStack [105] is an open source, rapidly growing, and frequently used *cloud operating system*, featuring a set of open source software for controlling a pool of resources in a private cloud. The resources can be managed and monitored via a web browser dashboard or an API. OpenStack has a modular architecture consisting of a set of *core* services for computing, networking, storage and identification, plus *optional* pluggable services for enhanced performance (e.g. containers or elastic map-reduce) [106]. The largest OpenStack cloud is operated by the CERN, with over 190k cores [107].

Hybrid Clouds

As their name suggests, hybrid clouds combine resources from both private and public clouds. The bond between these distinct infrastructures is generally not a feature of a cloud provider and has to be implemented through proprietary mechanisms. Scenarios that motivate the use of hybrid clouds include:

- Temporary bursts experimented in a private cloud with otherwise steady usage. In this case, additional resources need to be outsourced from a public cloud to guarantee that the service performance is attained during the burst.
- Organizations that use public clouds but where confidentiality agreements prevent them from transferring sensitive data off-premises and thus a secure private cloud is required as well to process these data.

2.2.3 Single- and Multisite Clouds

The facility where a cloud infrastructure is physically hosted is called a *datacenter* or cloud *site*. A datacenter is the largest building block in the cloud hierarchy; it is composed by a number of computing units (usually racks) interconnected through a set of switches using high speed networks. These computing units consist of tens to hundreds of multi-core nodes and their corresponding disk storage.

Figure 2.3 – Microsoft Azure's geo-distribution of their datacenters[2].

Single Datacenter

In single-site clouds, users leverage infrastructural advantages like *low-latency networks* and *data locality* to speedup computation, data retrieval and data storage. Additionally, cloud providers usually offer mechanisms to easily setup subnetworks of virtual machines within a single datacenter, which *facilitates the communication* among them. These benefits make single-site clouds to act like *pay-as-you-go supercomputers* in a way, what often motivates organizations to use them to execute large scale applications. In general, private clouds are deployed on single-site infrastructures.

Multiple Datacenters

In order to deliver a quality service for their users around the world, major cloud providers account for *multiple datacenters* scattered over the planet and organized in *geographic regions*. This set of geo-distributed datacenters serve the cloud for various purposes:

- Reduce service latency by fulfilling user requests from the closest datacenter to the user location.
- Guarantee secure data storage and availability through geo-redundant copies of the datasets.
- Provide tailored products in geographic areas with specific Internet regulations.

These service enhancements come at a cost, though. Applications that require multisite deployments should take several implications into account. Most of the communication between regions is done across the public Internet; therefore, users should apply appropriate encryption methods to protect their data. Also, since the network infrastructure is shared, the inter-site throughput turns both inconsistent along time and much lower compared to an intra-site network. Furthermore, local laws or business goals might prevent cloud providers from offering a particular service in a given region. This heterogeneity implies the overhead for users to test their developments in each target location.

[2] Image property of Microsoft
https://blogs.msdn.microsoft.com/uk_faculty_connection/2016/09/19/azure-data-centers-and-regions/.

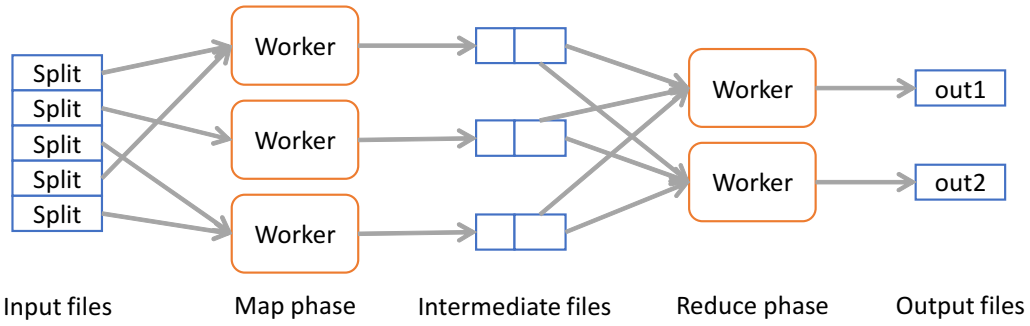


Figure 2.4 – The MapReduce programming model.

Each provider handles their datacenters in different ways to allow them to meet service or business goals. AWS reports having 16 regions, each subdivided in what they call *availability zones* (42 in total, consisting of one or many datacenters) [10]; these zones are interconnected by private fiber-optic networks for low latency and fault tolerance. Azure, on the other hand, is available in 34 regions around the world (Figure 2.3). The use of the India regions is limited to customers with business presence in the country. Azure services in Germany is provided under a data trustee model: customer data remains in Germany under the control of a third party [87].

Nowadays, many big data applications are deployed to multisite clouds, either to distribute parts of their execution closer to where the data resides, or because the capacity of a single site is not enough for their needs.

2.2.4 Data Processing on the Cloud: MapReduce

Cloud applications handling big volumes of data need to process them in parallel to optimize resource usage. These applications usually consist of a large number of computing threads running simultaneously in as many cores as there are available. A typical way to define such applications is by means of the well-known MapReduce model [39], introduced by Google in 2004. MapReduce hides the lower level details of parallelization, fault tolerance and load balancing, making applications simple to program, by means of a pair of methods applied to partitions of the input dataset.

The MapReduce Model

MapReduce is a highly-scalable programming model for parallel processing of very large datasets. Google's first implementation of MapReduce ran on clusters of commodity machines. As its name suggests, the model consists of two functions, *map* and *reduce*, which are defined by the user.

Map. The map function is applied to every input pair of data to produce one or several intermediate key/value pairs.

Reduce. The reduce takes from the intermediate pairs a set of values that share a common key, merges these values and produces (normally) a smaller set of values.

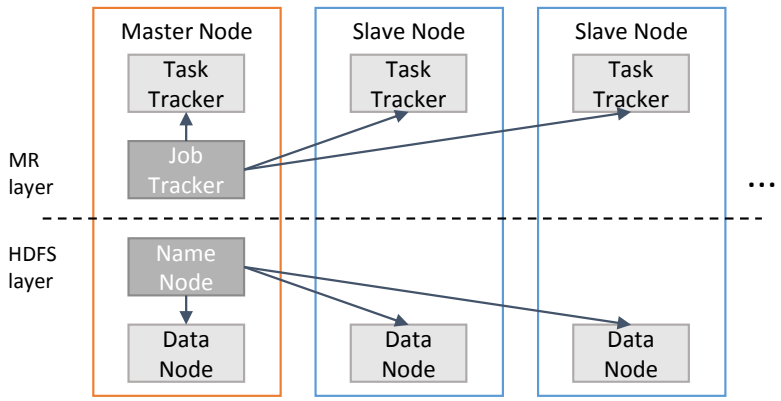


Figure 2.5 – A Hadoop cluster, divided into two logical layers: MapReduce and HDFS.

The regular flow of the MapReduce execution is as follows (Figure 2.4): (1) The input data are partitioned in many chunks. (2) The data are passed on to several *workers* which perform the *map* function on each data record. (3) The intermediate key/value pairs are shuffled by key and stored in intermediate files. (4) Pairs with the same key are passed to a *workers* which perform the *reduce* function. (5) The final partial results are grouped if needed.

MapReduce's scalability and facility to deploy in clusters of small machines (even PCs) have made it the most adopted model for processing large-scale applications. There are several implementations of MapReduce, the most popular being Apache Hadoop, since Google's own version was not publicly available.

Apache Hadoop

Hadoop [59] is an open source framework for large-scale computing. A Hadoop cluster consists of one *master* node and several *slave* nodes (Figure 2.5) that execute user-defined *jobs*. The original core components of Hadoop are a MapReduce engine and a distributed file system, HDFS [20].

The MapReduce layer is the execution engine, consisting of a single master *JobTracker* and one *TaskTracker* per node. The JobTracker splits the job in several tasks and schedules them to the TaskTrackers, monitoring and re-scheduling them in case of failure. The JobTracker provides mechanisms for location-aware optimization, assigning tasks to the closest node to where the data resides. The TaskTrackers are the actual executors of the map and reduce methods, they read data from and write data back to HDFS.

The HDFS layer handles file system metadata and application data separately. The master *NameNode* stores the namespace hierarchy, as well as attributes and permissions of files and directories. The data is split into blocks of customizable size, these blocks of data are replicated in several *DataNodes* (one per node). The mapping of block to DataNodes is also maintained at the NameNode. Read requests are managed by the NameNode, which determines the closest node to the requester that holds the required data block. To write a new block, the NameNode nominates a set of DataNodes and the block is written in a pipeline way.

The term *Hadoop* is loosely used to refer also to other Apache projects for scalable computing and storage linked to Hadoop, like Cassandra, Spark, Pig or Hive [59]. Hadoop YARN [138] is a more recent version of Hadoop, where job and resource management are decoupled. As mentioned in Section 2.2.2, fully-managed platforms to create Hadoop clusters are readily offered by the three major cloud providers. These platforms allow to easily manage and execute Hadoop jobs, and they include several of the projects from the Hadoop ecosystem.

Limitations

Despite the large number of big data applications that follow a MapReduce pattern, there are several scenarios in modern large-scale computing that the model can not handle properly. We identify two in particular: *multisite processing* and *application's expressiveness*.

Multisite Processing. As we previously stated, many applications are deployed to multisite clouds for several reasons (e.g. data proximity, insufficient resources). MapReduce was conceived to run on clusters of commodity hardware, in single-site infrastructures. Some implementations have managed to bridge MapReduce across multiple datacenters [141, 147]. However, we recall that MapReduce stores large intermediate files in disk between the map and reduce phases. In a multisite setting, where high-latency inter-site networks are the norm, writing and shuffling the key/value pairs for the reduce phase would incur very time-consuming data transfers. Additionally, the file system would stay in an inconsistent state during the periods where such transfers take place. As a result, the execution of multisite large-scale applications could actually be hampered rather than improved by MapReduce.

Application's Expressiveness. While the MapReduce model provides an scalable solution for numerous applications, it restricts the users to a pair of consecutive operations. Consequently, applications with complex dependency patterns are not possible to model as MapReduce jobs. For example, gather data from different sources, heterogeneous transformations running in parallel over the same data, or even applications where only a map or a reduce operation is necessary (without the burden of intermediate shuffling).

In order to overcome these limitations, a more abstract model for describing large-scale applications is becoming increasingly popular: *workflows*. Workflows are defined as loosely-coupled sequences of jobs, what allows them to be deployed to multiple datacenters.

2.3 Scientific Workflows on Clouds

A workflow is the automation of a multi-step business process in whole or in part [60], in which data are transferred and transformed along different stages called *jobs*. The concept of workflow emerged from business domains, as a model to define complicated processes. A business workflow is composed by several procedures linked together to achieve a business goal, passing information from one participant to the next. In an analogous way, the scientific community has adopted this model in order to face the challenge to represent and manage increasingly complex distributed scientific computations. This gave way to the emergence of *scientific workflows*.

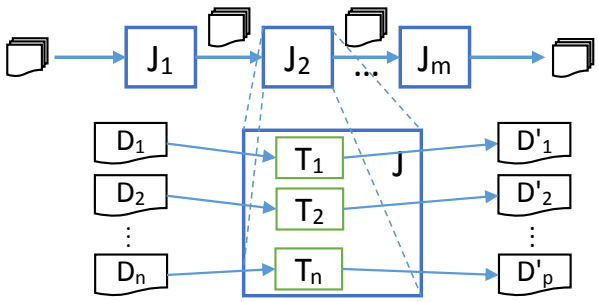


Figure 2.6 – Workflow *anatomy* depicting *jobs* (J), *tasks* (T) and *data chunks* (D).

2.3.1 Overview: The Scientific Workflow Model

A scientific workflow (SWf) involves individual data transformations and analysis steps, and mechanisms to link them according to the data dependencies among them [40]. As mentioned above, scientific workflows arose from the need to model complex, distributed applications. As a consequence, scientific workflow management typically involves dealing with the large volumes of data associated to such applications, as we recall from Section 2.1.1.

Business workflows are defined with a low level of abstraction. They allow to model event-based interactions, which trigger activities in response to some business action [74], e.g. send a confirmation email to the customer when a purchase has been successful. Their goal is to automate tasks to reduce human involvement in the process and reduce lags. In contrast, scientific workflows are conceived with a higher abstraction; each job represents a scientific process or computation. Scientists are provided with a workflow execution platform and a language (set of "rules") to define data dependencies between several applications. These platforms are called *Scientific Workflow Management Systems* (SWfMS), and are presented in Section 2.3.2.

Scientific workflows have enabled a new paradigm in collaborative research. Thanks to workflow definition rules and workflow management systems, data collected from diverse sources and distributed locations, transformed by different applications from various scientific domains, can now be combined and analyzed, leading to multidisciplinary science breakthroughs.

The focus of this thesis is on scientific workflow applications. Along the manuscript, the generic term "workflow" is used to refer to a scientific workflow. For practicality, the acronyms SWf and SWfMS are used in several sections.

Workflow Modelling

***Anatomy* of a workflow.** Scientific workflows are modelled as directed acyclic graphs (DAG), in which vertices represent data processing jobs and edges represent dependencies between them (Figure 2.6). A *job* (J) is the highest level of granularity. It depicts a piece of work (computation) that forms a logical step within a workflow representation. Workflow jobs may process multiple data fractions, depending on how the input data is split. Thus, one job can actually consist of several executable tasks for different pieces of input data. A *task* (T) is the representation of a job within a one-time execution of this job, which processes a data chunk (D), i.e. there is a task for each unit of data to be processed. Each task might (or

```
<workflow>
  <!-- Jobs -->
  <job id="job1" exec="foo.cc" />
  <job id="job2" exec="bar.jar" />
  <job id="job3" exec="merge.py" />
  <!-- Dependencies -->
  <child ref="job3">
    <parent ref="job1" />
    <parent ref="job2" />
  </child>
</workflow>
```

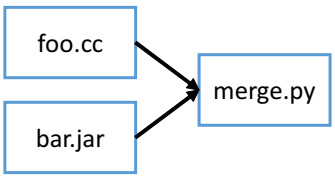


Figure 2.7 – A simple representation of a workflow (based on Pegasus DAX).

might not) output a new data chunk. The collection of output chunks form the dataset to be passed on to the next job.

Workflow representations. Scientific workflows are formally specified using different languages or building rules which are in most cases unique to a SWfMS. A common approach is to use XML documents with a system-specific schema [145], where each job and dependency is defined by a tag (Figure 2.7). Some systems also provide APIs to facilitate the workflow definition [42].

Each language has particular features that make them outstand. However, they still present some limitations for advanced users. For instance, they need to accommodate scientific process descriptions at multiple levels [40]: different scientists might have different degree of interest in a computation and required more in-detail workflow specifications. More importantly, a prevailing issue is the diversity of languages itself. The heterogeneity in workflow representation tools hinders cross-platform sharing and collaboration practices.

Core Workflow Features

Several characteristics define and are common to most of the scientific workflows. The features listed hereunder are a representative sample of them, which motivated part of our work during this thesis.

Common control-flow patterns. While some authors provide an extensive list of specific job dependency patterns [121]; in general, all of them agree that at least there are four basic flow patterns between workflow jobs [111, 101]. Further studies [127] show that the workflow applications are typically a combination of these patterns.

Pipeline. Also known as *sequential* or *serial* execution. Means that the execution of a job is enabled after the execution of a single preceding job.

Scatter. Called also *parallel split*, or *broadcast*. Indicates the divergence of a branch into two or more parallel branches that can run concurrently.

Gather. *join* or *synchronization*. Represents the convergence of two or more branches into one. The job where all the branches gather, cannot begin its execution until all the previous have finished.

Parallel. Means that two jobs do not actually have data dependencies and therefore can be executed in parallel. The possibility to specify this pattern is not available in every SWf definition language.

Provenance management. Provenance is the *metadata* that captures the derivation history of a dataset [70], including the original source, intermediate data, and the workflow steps that were applied to them. It is used for workflow analysis and reproducibility. The number of provenance parameters recorded and the way they are handled vary from system to system, but every SWfMS keeps track of provenance metadata. Taverna, for example, stores it as a graph [15], and distinguishes between process-related and data-related provenance. On the other hand, MTCProv [51] records domain-specific provenance in the Swift SWfMS.

Many small files. Workflows generate and access a huge number of relatively small files [25]. This follows the fact that each job is parallelized into many tasks, which process a fraction of the data. The natural question is: *what is a small file?* Several scientific domains such as climatology, astronomy, and biology generate data sets that are most conveniently stored in small files: 20 million images hosted by the Sloan Digital Sky Survey with an average size of less than 1 MB [97], up to 30 million files averaging 190 KB generated by sequencing the human genome [19].

Write once, read many times. During an experiment, workflow applications typically generate a large number of files, that are no longer updated given the acyclic quality of scientific workflows. Instead, these files can be read many times by subsequent tasks when a scatter pattern is defined. This characteristic allows to relax concurrency conditions: after a file or data chunk has been created, it can be accessed simultaneously.

Porting Workflows to Clouds

Many existing workflows were conceived to run on top of HPC systems, they are normally deployed to a cluster of nodes where non-sequential jobs run concurrently, and use parallel file systems to exchange data between jobs in different nodes [64].

In order to leverage the cloud-specific characteristics, workflows have been gradually ported to the cloud. As opposed to other infrastructures like clusters or grids, clouds offer the possibility to provision additional resources on demand. Whenever the number of workflow jobs exceed the number of available resources, new nodes could be provisioned so that parallel jobs could be executed simultaneously, avoiding queuing delays. In addition, cloud virtualization allows platform-dependent workflows to be executed regardless of the underlying infrastructure. Nevertheless, bringing workflows to clouds involves several adaptations.

As we have seen previously on this chapter, clouds have emerged as convenient infrastructures to host large-scale applications, and workflows are not the exception. With the

right configuration, a cloud environment can be adapted to behave, to some extent, as an HPC infrastructure (Section 2.2.3). First, clearly all the cloud nodes have to be deployed to the same datacenter, so that network latencies are low. Also, the cloud provider should allow to group the nodes in a subnetwork which enables a *virtual cluster* [50]. In order to expand their capabilities (and market) to host cluster-based applications, some public clouds have implemented solutions for scalable, virtual HPC clusters [88, 113].

Additionally, we recall that workflow jobs communicate through storage systems, usually an HPC parallel file system, such as Lustre [104] or GPFS [125]. Cloud storage is generally based on services managed independently of the nodes, like S3 [8]. Using this kind of external storage would turn the workflow highly inefficient, since a large number of files would be taken out and into the cluster. A reasonable workaround is to deploy a virtual file system within the cluster, for instance PVFS [24].

The cloud-workflow adaptation effort has been done also in the other direction. Some SWfMS have bridged this gap and have developed cloud-enabled versions of their systems, for instance Pegasus [42] and SciCumulus [103], so that the migration to the cloud is done transparently and effortlessly for their users.

Workflow Limitations on Single-Site Clouds

The adaptation of workflows to the cloud discussed above corresponds to single-site clouds. Enabling cloud workflows represents a step forward in terms of portability and scalability. Nevertheless, executing large-scale scientific workflows on a single datacenter has its own limitations.

Datacenter capacity. Workflows operate large volumes of data. In a cloud workflow, fractions of these data are stored locally on each node as part of a virtual file system. In order to efficiently process those data, hundreds of task must run concurrently in the virtual cluster, what requires a very large computing and storage capacity. However, clouds establish usage quotas per resource and/or per datacenter [89, 58]. These quotas ensure a fair use of the resources, protect users from unexpected spikes in usage, and are set according to the client's financial capabilities. As a result, large-scale workflows might easily exceed their resource allocation in an single datacenter.

Data dispersion. Workflows have enabled multidisciplinary collaborations, this often means that the jobs that integrate them are executed in different institutions far away from each other. Moving data from their original location to a single host datacenter might be costly or even represent a security risk. There are situations where data must be processed in a specific location due to government regulations; we recall the case of Azure Germany, where all user data stays in the country to meet EU data protection requirements [87]. Additionally, even data from a single organization might be originated from distributed sources: Facebook contents generated all around the world about a certain topic are stored in the closest datacenter from where it originated.

In both cases, there is a motivated need to deploy a workflow to more than one datacenter. Whether to meet the required computing or storage power, or to process data closer to where

it resides, multisite scientific workflows stand out as a solution that enables to balance performance and cost as they execute. Generally, these multisite workflows are hosted in public clouds, which provide a larger number of widely distributed datacenters.

It is evident that the functionality and efficiency of an HPC infrastructure cannot be mimicked across geo-distributed datacenters. Still, for the aforementioned situations, multisite workflows are likely the best, if not the only solution.

2.3.2 Scientific Workflows on Clouds: State of the Art

In this section we briefly summarize some of the most popular workflow management systems, with a focus on *scientific* WfMS. We then discuss their limitations in order to highlight the challenges that we address in this thesis.

Workflow Management Systems

Workflow management systems (WfMS) are platforms that provide a set of tools to configure, run, and monitor workflows.

Commercial WfMS. There are several proprietary workflow management tools for business. Their main objective is to turn repetitive tasks into multi-step workflows. They allow to create data flows between commercial applications to receive notifications, synchronize devices or collect data. Examples of these applications are Microsoft Flow [90] and SAP Business Workflow [74]. While these business platforms reduce overhead in everyday tasks, their capabilities are limited:

- They do not support large-scale processes.
- They offer a limited number of building blocks (usually proprietary applications).
- There is no room for user defined blocks, what greatly limits their target market.

Commercial WfMS are widely used for business processes nowadays; however, they significantly differ from scientific workflows in their purpose, degree of autonomy and volumes of data, as we have mentioned before.

Scientific WfMS. Scientific workflow management systems (SWfMS) are used in many science fields nowadays, from bioinformatics, to astronomy and social sciences. They coordinate the operation of every component of a workflow at different levels of granularity (e.g. scheduling a full job to a specific execution node, or re-executing individual tasks that failed). They incorporate simple rules and syntaxes to define workflow jobs and data dependencies, so that the transitions between jobs are fully automated and users can focus on extracting value from their data. Most SWfMS offer generic services like provenance metadata management, and provide a language for workflow definition. However, every SWfMS has its own specificities; we list a selection of systems and their key features hereunder. While all of them were originally designed for running on HPC infrastructures, they all support cloud execution today.

Pegasus. Pegasus [42] is a platform to execute workflow applications in different environments, from desktop machines to clouds. Although initially not conceived for clouds,

Pegasus now runs on top of infrastructures like Amazon EC2, Google Cloud and Chameleon. Workflows in Pegasus are described using DAX language (DAGs in XML), that can be generated manually or through a simple multi-language API. Pegasus outstands for its *robust error-recovery* capabilities: upon an error, Pegasus tries to recover by retrying tasks or the entire workflow, by providing workflow-level checkpointing, by re-mapping portions of the workflow, by trying alternative data sources for staging data, and, ultimately, by providing a rescue workflow containing a description of only the work that remains to be done [110]. In terms of execution, Pegasus features *storage clean-up* as the workflow is executed, in order to free space to enable data-intensive workflows to run on storage-constrained resources. Pegasus provides automatic runtime provenance storage and several tools for *provenance analytics* such as pegasus-statistics and pegasus-plots.

Swift. Swift [143] is a system for reliable specification, execution, and management of large-scale workflows. Swift supports multi-task applications, where tasks are coupled by disk-resident datasets. Swift combines a scripting language for high-level specification of complex computing processes, and an execution engine capable of dispatching millions of tasks running on supercomputers or even *multisite grids*. Swift scripts use a C-like syntax to describe data, application components and their relations. They are written as a set of composed functions. Atomic functions specify the execution of external programs. High-level functions are composed as graphs of subfunctions. In Swift, the structure of a workflow can be constructed and *expanded dynamically*. Swift introduced *mapped types*, which represent data residing in a file, they can be seen as pointers to files and they can be used to identify files that have to be transferred to remote sites or passed to applications. Swift provides a step-by-step tutorial for deployment on clouds [131].

Taverna. Taverna [145] is an open-source and domain-independent SWfMS. It provides a workbench for *fully graphical workflow design* with drag-and drop-components that allows easy creation of new workflows or load and modification of existing ones. Semantically annotated components can be shared among workflows. It also features a wide range of tools for workflow execution, including: failover and retry, customizable job looping, intermediate data viewing, workflow debugging, pause/resume or cancel the execution. One Taverna's key characteristics is that it allows to interact with a running workflow through an *interaction service*. Once the service is invoked from within the workflow, it uses a web browser for interaction. Taverna keeps track of workflow data and execution provenance. It does not manage provenance of workflow editing (versioning control); however, it does keep track of workflow definition metadata in terms of authorship [4]. Projects for next-generation sequencing have been run using Taverna and Amazon EC2 Cloud [13]. Taverna has recently been accepted and is currently transitioning to the Apache incubator.

Chiron. Chiron execution engine [100] is implemented in Java, and was also designed for HPC environments. It integrates PostgreSQL to provide reliable execution management. Chiron stores control and provenance metadata on the database system. A particularity of Chiron engine is accounts for six workflow *algebraic operators*, which allow for optimization, dynamic scheduling and runtime workflow steering. The operators are Map, SplitMap, Reduce, Filter, SRQuery and JoinQuery. The last two apply

relational algebra queries directly on the database. Chiron extensions include SciCumulus [103], an encapsulation of Chiron to the cloud, which implements *elasticity* and adaptivity according to the workflow execution and the current state of the environment. Most notably, another extension is Multisite Chiron, a *multisite manager* running on geo-distributed clouds like Microsoft Azure [72]. Chiron is the base SWfMS used in this thesis.

Limitations

Scientific workflow management systems were not intended for cloud infrastructures, much less for multisite clouds. The above systems are examples of a continuous effort to port workflows to clouds; still, most of them concern only single-site architectures. SWfMS leverage HPC-like cloud infrastructures, which are generally virtual clusters with a shared file system. However, workflow data are often so huge and widespread that they need to be distributed to many cloud datacenters for storage and processing, as previously exposed. Multisite workflow execution brings some limitations into play.

No shared-data system. The first limitation is directly related to the workflow model. Workflows communicate through files that one job writes to the file system so that the next job can read them. There are no (virtual) clusters or local networks across datacenters, and so parallel or distributed file systems cannot be used. The closest approach to a shared storage is to use the cloud's provider global storage (e.g. S3 [8] or Azure Blobs [85]) which are usually accessible via URIs or APIs. This solution, however, requires to change the logic of the workflow, add external libraries to connect to the storage service and make a two-step operation to write or read a file, adding significant overhead. First, the file is created in the local file system of the node, then it is transferred to the storage service (similarly for file reading). Other possibility is directly transfer files through standard protocols like SCP, but this would require additional effort from the developers to take care of the full transfer process (including fault tolerance). Moreover, inter-site data transfers are billed.

High-latency networks. If handling file transfers were not enough, multisite management also implies that these data transfers have to be done through the high-latency networks that interconnect datacenters. Inter-site communication (from control messages to big file transfers) usually happens via the public Internet. This has the "usual" implications: an increased latency due to the shared network combined with the physical distance, an unstable throughput depending on the concurrent usage, and the possible loss, theft or corruption of the data. To cope with this limitation, public cloud providers provide some dedicated, secure, high-speed inter-site networks. AWS connects its *availability zones* within a region with fiber-optic links [10], but this is not possible across regions. Azure offers ExpressRoute [91], a service for private connections bypassing the public Internet. A fixed monthly fee for the connection plus additional fees per gigabyte for outbound transfers are charged, though.

Centralized management. Workflow management systems, usually include a master node from where all the execution is orchestrated. Jobs are scheduled, the execution is monitored and provenance and other metadata are collected. In a single-site infrastructure, latencies

are negligible and having a centralized management of these aspects might not impact the total makespan. On the other hand, handling a multisite execution in a centralized node can become a bottleneck. Control messages and metadata updates have to traverse low-speed networks; what used to take microseconds might take orders of magnitude longer across datacenters (see Figure 3.1). Task-data co-scheduling turns inefficient: two co-located jobs sharing file in the same datacenter, might need to query a distant centralized metadata server to find out the file location. Furthermore, with the very large number of files handled by today's workflows, a centralized management of the execution, data, and provenance, the workflow performance could degrade even in a single-site cluster.

2.4 Challenges Addressed

Some SWfMS support multisite workflow processing to the best of their capabilities. Pegasus [33] proposes three heuristics for partitioning the workflow in sub-workflows considering storage constraints. Multisite Chiron [72] is a more recent approach which in addition to scheduling adapts the provenance model to a multisite environment. Nevertheless, there are several challenges to multisite workflow execution that remain unaddressed. This thesis focuses on two of them, as described next.

2.4.1 Metadata Management

A workflow's execution generates a significant amount of metadata, e.g. scheduling metadata (i.e. which task is executed where), data-to-task mappings, and file metadata. Most of today's SWfMS handle metadata in a centralized way. File-specific metadata is stored in a centralized server, either own-managed or through an underlying file system, while execution-specific metadata is normally kept in the execution's master entity. Controlling and combining all these sorts of metadata translate into a critical workload as scientific datasets get larger. The CyberShake workflow, for instance, runs more than 800 000 tasks, handling an equal number of individual data pieces, processing and aggregating over 80 000 input files (200 TB) [63]. With many tasks' runtime in the order of milliseconds, the load of parallel metadata operations becomes very heavy, and handling it in a centralized fashion represents a serious performance bottleneck.

Managing metadata in a centralized way for such scenarios is not appropriate. On top of the congestion generated by concurrent metadata operations, remote inter-site operations cause severe delays in the execution. To address this issue, some file systems rely on the use of decentralized metadata servers [132].

State-of-the-art SWfMS have put relatively low attention to their metadata handling strategies, even in a single-site setting. Particularly, *multisite workflow metadata management support is seldom addressed, if at all*; this is likely due to the lack of solutions for multisite workflow execution, in general.

- Pegasus metadata are stored in the same database as workflow data [112]. Concurrent access of the same database for data and metadata might yield very poor performance, especially in multisite clouds. Our goal is to *optimize metadata access through multiple decentralized metadata servers*.

- Taverna considers metadata "*a first class citizen*" and distinguishes between workflow specification metadata and workflow execution metadata (provenance) [15]. It also provides services for maintenance and curation of metadata. Our interest is not to curate the metadata, but rather to *bring it as early as possible to where it is needed*.

We claim that efficient, decentralized metadata management significantly improves the performance in a multisite workflow execution. Chapters 3 and 4 propose two different techniques to achieve it: a hybrid distributed/replicated metadata registry, and selective handling of relevant metadata. This is the main challenge targeted in our thesis.

2.4.2 Elastic Usage of Resources

Elasticity is one of the driving features that brought workflows to cloud environments. The possibility to enhance virtual clusters with on-demand resource (de)provisioning allows workflows to expand beyond their original limits, without incurring much larger investments. Several studies contemplate the elasticity component as a design principle. In [120], the authors propose an algorithm for resource provisioning and scheduling for workflows, accounting for cloud elasticity and heterogeneity under deadline constraints. In [146], excessive workers are removed to meet the master's capacity and avoid resource waste. Nagavaram et al. [95] use Pegasus for dynamic allocation for a large-scale biology application. All previous cases provide what we could call *classic* elastic scaling: new virtual machines are added to meet a growing demand for resources.

In Chapter 5, *we present a study of application parameters that steer elasticity*, towards the implementation of a system for *smart* elastic scaling for workflows. Smart scaling aims not only to determine *when* to provision, but also *which* resource to instantiate (in terms of size, configuration and location), according to the current and the foreseen state of an application. We believe that smart scaling can optimize the resource usage in a cloud, while improving the workflow execution. The study was carried out in collaboration with ANL.[3]

[3] Argonne National Laboratory, USA.

Part II

Contributions

Chapter 3

Metadata Management for Geo-Distributed Clouds

Contents

3.1 **Problem: Huge Latency for Remote Metadata Access** 33
3.2 **Solution: Adaptive Decentralized Metadata Handling** 35
3.3 **Design Principles** 35
3.4 **Strategies for Multisite Metadata Management** 38
3.5 **Discussion** 41
3.6 **Conclusion** 43

THE cloud *site* or *datacenter* is the largest building block of a cloud. It contains a broad number of interconnected compute and storage nodes, providing computational power available for rent. Clouds, particularly public ones, are generally made up of several sites scattered over the planet. An application that is deployed across several of these datacenters is known as *multisite application*.

Large-scale workflows benefit from multisite distribution, as they can aggregate resources beyond the limit of a single datacenter. Besides the need for additional compute resources, workflows have to comply with several cloud providers requirements, which force them to be deployed on geographically distributed sites. For business safety and load balancing, public clouds impose limits of maximum cores or virtual machines per user within a datacenter [89]; any application requiring more compute power likely needs to be distributed across several sites.

3.1 Problem: Huge Latency for Remote Metadata Access

Metadata are now bulkier, more complex, but also more relevant in any large-scale application; and workflow metadata are not the exception. Indeed, the size of the metadata can

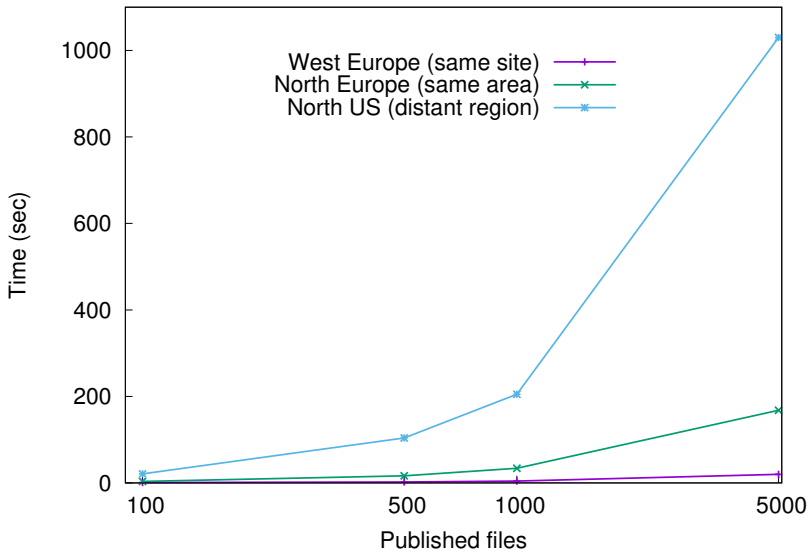


Figure 3.1 – Average time for file-posting metadata operations performed from a node the West Europe datacenter. The metadata server is respectively located within the same datacenter, the same geographical area and a remote region (log scale).

be proportionally large for a small file in a workflow. The shift towards multisite workflow processing is greatly handicapped by the limited metadata support for such scenarios. Most files in even the largest workflows are small, with median file sizes in the orders of kilo- or megabytes, yet generated in *large numbers*. This means that metadata access has a high impact (sometimes being dominant) on the overall workflow I/O. Conventional cluster file systems are optimized mainly for scaling the data path and lack proper support for the geographical distribution of the metadata. In such cases, the common approach is to use a single metadata server or a federation of metadata servers within a single site, serving the whole multisite application. Even VMs accessing data on their local datacenter may need to yield expensive remote calls to the datacenter where the metadata registries are located. Users have to set up their own tools to move metadata between deployments, through direct endpoint to endpoint communication (e.g. GridFTP [2], scp, etc.). This baseline option is relatively simple to set in place, using the public endpoint provided for each deployment.

The major drawback in this setting is the high latency between sites. The numerous metadata requests have to traverse the slow WANs connecting the datacenters (which in most cases are the property of the ISPs, so out of the control of the cloud providers), limiting drastically the achieved throughput of the workflow. A simple experiment conducted on the Azure cloud and isolating the metadata access times for up to 5 000 files (Figure 3.1) confirms that remote metadata operations take orders of magnitude longer than local ones. This has a high impact on the overall workflow makespan, particularly for workflows handling many small files, when the metadata operations are very frequent. Clearly, this paradigm shift calls for appropriate metadata management tools, that build on a consistent, global view of the entire distributed datacenter environment.

3.2 Solution: Adaptive Decentralized Metadata Handling

> *❝ We propose to evaluate different metadata handling strategies, that result from exercising distribution and replication techniques across the available datacenters. ❞*

Workflows contain a variety of job relations, that can be as simple as a sequence of data dependencies between jobs, or as complex as a mixture of multiple inputs, parallel jobs and multiple outputs. However, some common data access patterns can be identified from these graphs: pipeline, gather, scatter, and parallel, as described in Section 2.3.1; workflow applications are typically a combination of them.

Given the unlimited possible combinations of data access patterns, a single approach to mitigate the potential metadata bottleneck in multisite environments will certainly not fit all. Therefore, we propose to evaluate different metadata handling strategies, that result from exercising distribution and replication techniques across the available datacenters. In the next section we introduce our design principles, driven by recent workflow workload studies on traces from several applications domains. We opt to keep all *metadata in memory*, in a uniform *DHT based cache*, distributed across cloud datacenters and following a *2-level hierarchy*: metadata are first partitioned to the datacenters where they are likely to be used (leveraging information from the workflow execution engine) and then replicated to other datacenters.

We analyze four configurations for improving concurrent metadata and workflow file I/O performance in multisite clouds. We characterize each of these approaches according to their performance with a given workflow pattern in order to design a lightweight and scalable geographically distributed metadata service. The strategies include:

- A baseline setup with a *centralized* server.
- A distributed layout with metadata *replicated on each site*.
- A set of *decentralized* handlers with *no replication* on the metadata.
- A hybrid approach with *decentralized* handlers, adding *local replication*.

Out of these strategies, the first one has been previously applied to separate workflow engine implementations [100] and therefore we use it as a baseline. The remaining three are novel proposals in the context of geo-distributed workflows. We discuss all of them in section 3.4.

3.3 Design Principles

In this section we delve into the design space of our proposed strategies and audit the trade-offs between different design decisions. In essence, our proposal consists of a hierarchical metadata partitioning in order to hide the latency and reduce I/O through three simple strategies: full replication, full distribution and an intermediate, hybrid approach.

Hybrid Distributed/Replicated DHT Based Architecture

Keeping workflow metadata on a single server is straightforward to implement but the large number of requests in a typical scientific workflow would easily saturate its resources. Distributing the metadata to multiple nodes seems the natural approach, yet the non-trivial

endeavor is how to make this distribution. Most existing works on metadata partitioning simply consider the namespace partitioning and the distribution of shares to servers (as detailed in Section 2.3.2). Such pure partitioning approaches may bring potential performance and scalability problems when used in geo-distributed scenarios, like multisite clouds. For instance, it is difficult to apply updates, since these may incur costly communications among geographically distributed servers.

We propose a hybrid approach mixing distribution and replication of metadata to address these problems. First, we divide the metadata servers into datacenters based on their location information. Within each datacenter, metadata are distributed to the servers, then it is replicated to different datacenters. With this approach, updates can be applied by only updating shares in one datacenter and propagating them to other datacenters. However, distributing metadata into multiple physical nodes poses new challenges such as load balance and fault tolerance. In this setting and depending on the data structures used (e.g. trees, hashes), the time complexity for search operations can vary drastically. We argue that hashing is a good option for metadata scattering as distributed hash tables (DHT) [52] have proved to be highly scalable in practice with a constant-time query cost. Also, a flat namespace implemented by hash tables exposes a simpler and less error-prone interface avoiding expensive operations required by others data structures (e.g. distributed trees). Thus, the hybrid approach in conjunction with the DHT can significantly reduce the user perceived response latency for update accesses.

Uniform In-Memory Caching

A key observation from the workflow traces is that traditional workflow metadata design incurs an excessive number of disk operations because of metadata lookups: the file's metadata must be read from disk into memory in order to find the file itself. While insignificant at a small scale, multiplied over millions of small files, using disk I/O for metadata is a limiting factor for read throughput.

We therefore opted to keep all metadata in memory, which we make practical by reducing the per file metadata. That is we only store the information necessary to locate files and we don't keep additional POSIX type metadata, like permissions, since they are normally not used in a scientific workflow, i.e. during the workflow execution the files produced are used by the same user(s). To store the metadata, we rely on a dedicated cache, uniformly distributed across all workflow's datacenters. As opposed to existing distributed caches (e.g. Memcached [49]) which aggregate memory resources from the actual deployment of the application, we argue in favor of a separate cache layer. This allows the data tier to scale independently and guarantees non-intrusiveness on the workflow execution. Our standard cache tier provides high availability by having a primary and a replica cache. If a failure occurs with the primary cache, the replica cache is automatically promoted to primary and a new replica is created and populated.

Leverage Workflow Metadata for Data Provisioning

Adequate *data provisioning* is crucial when scientific workflows run on geographically distributed clouds. A task might require a very large file stored at a distant location. If the data are not in place, idle execution times occur, impacting the workflow's makespan. It is therefore essential to know in advance *what* data would be needed, *when* and *where*.

Workflow execution engines exploit the workflow's metadata (i.e. data provenance, data dependencies between tasks) for smarter task scheduling strategies [100]. We claim that a similar approach can be adopted to optimize data provisioning. By efficiently querying the workflow's metadata, we can obtain information about data location and data dependencies which allow to proactively move data between nodes in distant datacenters before it is needed, keeping idle times as low as possible. In this contribution we study the first step towards such optimizations: a reliable metadata management service that ensures that metadata operations are also carried out efficiently across multisite clouds.

Eventual Consistency for Geo-Distributed Metadata

Scientific applications running in a single site benefit from infrastructural elements - such as high speed communication channels, shared memory or multicore architectures - to improve execution time and optimize resource usage while ensuring a consistent state at any given time. Part of this optimization concerns metadata operations. Today, metadata updates and lookups have been improved up to take a few microseconds, with middleware able to carry out hundreds of thousands of metadata operations per second [119]. Unfortunately, in a multisite cloud, said elements are usually not present and thus even simple metadata operations might take long time to propagate, particularly when the datacenters are geographically distant. In order to maintain a fully consistent state of the system, all nodes would have to wait until the newest operations are acknowledged by every member of the network. This is evidently inefficient considering the potentially long (physical and logical) distance between instances and the large number of metadata operations generally performed.

Therefore we argue for a system where every metadata update is guaranteed to be *eventually* successful, but clearly not in real-time. That is, rather than using file-level eager metadata updates across datacenters, we favor the creation of batches of updates for multiple files. We denote this approach *lazy metadata updates*: it achieves low user-perceived response latency and high scalability in a widely distributed environment by asynchronously propagating metadata updates to all replicas after the updates are performed on one replica. Yet, this lazy approach only guarantees eventual consistency, meaning that if any other tasks try to access the distant metadata at the same time, the result will be undefined.

However, eventual consistency is perfectly in line with the observations from the workflows traces. The typical data access pattern is write once/read many times, with readings occurring in two situations. For intermediate results, data are used as input for the next task(s) in the workflow, but in these cases the engine scheduler takes care to schedule the task close to the data production nodes (i.e. in the same datacenter) so the metadata updates are instantly visible here. For final results, data might be accessed from remote locations, but typically this a posteriori analysis takes places long after the workflow execution has finished, leaving enough time for the lazy updates of the metadata to propagate. So, in both cases, the eventual consistency is not affecting the application performance or coherence.

Eventual consistency is, in practice, the default consistency model for multisite distributed systems (e.g. Facebook's graph store TAO [22]), where machine failures and network partitioning are a virtual certainty.

3.4 Strategies for Multisite Metadata Management

In the remainder of this manuscript we identify as *metadata registry* the instance or set of distributed instances in charge of managing metadata entries (shown as red diamonds in Figure 3.2). Every metadata registry instance is reachable by every node in the network. For the sake of practicality, we denote as a *read* the action of querying the metadata registry for an entry, and as a *write* the publishing of a new entry. Note that since a metadata entry can be created by one node and subsequently updated by others, a *write* operation actually consists of a look-up read operation to verify whether the entry already exists, followed by the actual write. File metadata entries are stored following a *key-value* approach, where the key is determined by a unique name of a file and the value indicates the node(s) where it is located.

We analyze the impact of high latencies as a consequence of the physical distance between an execution node and the corresponding metadata registry instance to/from where it will write/read an entry. The choice of an instance will depend on the management approach that we select; such approaches are described later in this section. Independently of the approach, in the following we use the following terms to qualify physical distance between a node and a metadata registry:

a) local — the node and the metadata registry are in the same datacenter;
b) same area — the node and the registry are in different datacenters of the same geographic area (e.g. both datacenters are located in Europe);
c) geo-distant — the datacenters are in different geographic areas (e.g. one in Europe, the other one in the US).

Both *b)* and *c)* situations can also be referred to as *remote*. Our design accounts for several datacenters in various geographic regions in order to cover all these scenarios. We have focused on four metadata management strategies, detailed below and depicted in Figure 3.2. In all cases, each datacenter is represented by a gray box, which contains a number of execution nodes (orange circles) and may contain an instance of the metadata registry as well (red diamonds). Solid lines connecting nodes and metadata registries denote metadata operations (reads or writes). The dashed lines represent a very large physical distance between datacenters; the ones on the "same side" of the line fit the *same area* scenario, whereas datacenters on "different sides" are *geo-distant*.

3.4.1 Centralized Metadata (Baseline)

In workflow management systems like Pegasus, metadata are managed and stored in the same database as the workflow data itself [112]. Even popular specialized distributed file systems rely on a centralized metadata server. In HDFS [20], for instance, in order to simplify the architecture of the system, all metadata handling is left to a single *NameNode*. As opposed to Pegasus approach, this dedicated node does not store any user data.

Following HDFS architecture, we first consider a single-site, single-instance metadata registry, independent of the execution nodes, arbitrarily placed in any of the datacenters (Figure 3.2a), which will serve as a *state-of-the-art baseline*. In this setup, the application processes are run on nodes which are distributed both locally and remotely with respect to the site of the metadata registry. In the case of non-local accesses to the centralized metadata, high-latency operations may occur. While a centralized server guarantees a higher level of

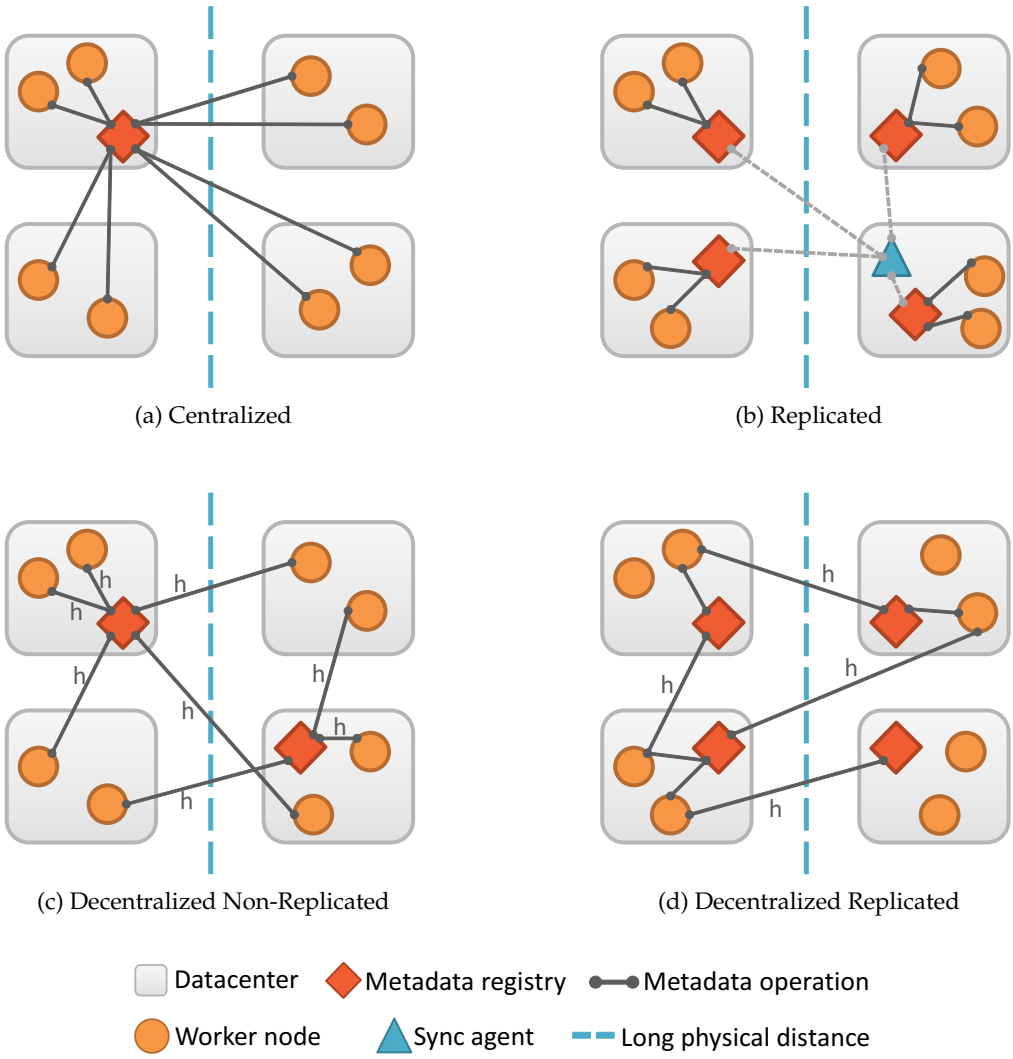


Figure 3.2 – Strategies for geographically distributed metadata management.

metadata consistency, it can quickly turn into a bottleneck as the workload increases. The purpose of this approach is to establish the (low) threshold at which these bottlenecks develop.

3.4.2 Replicated Metadata (On Each Site)

Our second strategy builds on the observation that local metadata operations are naturally faster than remote ones. Given a set of distributed datacenters, we place a local metadata registry instance in each of them, so that every node can locally perform its metadata operations. At this point, metadata information is processed quickly, but it would only be known at local level. Then, we propose to use a *synchronization agent*, a worker node whose sole task is to replicate across the network the content of the local metadata registries.

The synchronization agent systematically queries all registry instances for updates, and

leverages non-metadata-intensive periods to perform batch synchronizations in all metadata registries. In this way, neither the execution nodes nor individual metadata registries are concerned with remote operations. The strategy is depicted in Figure 3.2b: the synchronization agent is presented as a blue triangle and can be placed in any site. The dotted lines between the agent and the registry instances represent the synchronization communication.

3.4.3 Decentralized, Non-Replicated Metadata

In our previous *replicated* approach, we took advantage of non-metadata-intensive computation time to maintain a synchronized distributed metadata registry. Even if a metadata registry instance is locally deployed in each datacenter, this strategy is still centralized in that it relies on a single synchronization agent, which can become a potential bottleneck, particularly in the case of a metadata-intensive workflow. Even a multi-threaded synchronization agent might not provide a sufficiently fast synchronization to keep resource-waiting idle time at its lowest. Taking this into consideration, our third strategy favors *decentralization*, based on a distributed hash table (DHT). We maintain an instance of the metadata registry in each of the active sites. Every time a new entry is written to the metadata registry, we apply a hash function to a distinctive attribute of the entry (e.g. the file name) to determine the site where the entry should be stored by computing a simple modulo operation between the obtained hash value and the number of available sites, the hashing is indicated by an h in Figure 3.2c. A similar procedure applies for read operations to identify the metadata registry instance in charge of a given entry. Note that in this case the metadata are partitioned across the registry instances, so the contents of these instances are no longer identical in this strategy: each instance stores a share of the global set of metadata entries.

This approach involves remote operations: on average only $1/n$ of the operations would be local, where n is the number of sites. However, we notice two main improvements with respect to a centralized approach. First, as the registry is now distributed, metadata management is done in parallel by several instances, dividing the number of operations per instance and reducing the metadata-related idle time per node. Second, hash functions guarantee that identical input strings will always produce identical hash values; hence, we can consistently determine the location of an entry from the hash value of its distinctive attribute. In this way, read operations require a single, direct lookup in a specific site, thus metadata operations remain of linear order, even when the registry is distributed.

3.4.4 Decentralized Metadata with Local Replication

Our last proposal aims at further leveraging the distributed setting of the metadata registry described above. As observed in Figure 3.1, local metadata operations take negligible time in comparison with remote ones, especially when the total number of operations becomes large, which is the case of data-intensive applications. Keeping this in mind, we propose to enhance the DHT-like approach with a local replica for each entry (Figure 3.2d).

Every time a new metadata entry is created, it is first stored in the local registry instance. Then, its hash value h is computed and the entry is stored in its corresponding remote site. When h corresponds to the local site, the metadata are not further replicated to another site but will only stay at that single location. For read operations we propose a two-step hierarchical procedure: when a node performs a read, we first look for the entry in the local metadata registry instance; with local replication, assuming uniform metadata creation across the

(a) Without local replication

(b) With local replication

Figure 3.3 – Decentralized metadata: local replicas avoid costly remote read operations.

sites, we have twice the probability to find it locally than with the *non-replicated* approach. Only if the entry is not available locally, it is then searched for in its remote location, determined by its hash value. Compared to the previous scheme (without replication), we expect that, overall, the gain (in terms of latency and bandwidth) due to an enhanced probability to successfully look up metadata locally will be higher than the extra overhead added by local replication to the previous scheme.

To illustrate the benefits of local replication, we take the following interaction example involving two nodes n_1 and n_2 running in the same site s_1: n_1 writes an entry to the metadata registry, read by n_2, the location of the entry being determined by a hash function. Assume that the hash value places the entry in a geo-distant site s_2. Two situations may occur:

- In the *non-replicated* approach, both read and write operations would be remote and take up to 50x longer than a local operation (Figure 3.3a).
- With *local replication*, the write operation keeps a local copy and the subsequent read is performed locally, saving one costly remote operation and making reads up to 50x faster (Figure 3.3b).

3.5 Discussion

In this section, we address two questions that arise from applying our strategies to different multisite workflows: first, *which of the proposed strategies works better for each type of workflow?* Then, *how is a workflow actually partitioned for multisite execution?*

3.5.1 Matching Strategies to Workflow Patterns

Following the previous design considerations and the various existing workflow patterns, it is expected that no single hybrid strategy will outperform the rest. For instance, we have witnessed from our preliminary tests (as shown in Figure 3.1) that a centralized approach performs just as well as decentralization when a workflow operates at small scale, i.e. using few nodes, managing at most 500 files each, running in a single site. Low latencies of intra-datacenter transfers coupled with the proximity of metadata servers enable a high throughput and reduce the access time. In such a scenario, the effort of putting in place a distributed metadata handling mechanism is likely not worth it. Therefore, we pose the question of *what strategy would best match what type of workflow?* To answer it, we reason about the common workflow characteristics (Section 2.3.1) and hypothesize the following situations, that are experimentally validated in Chapter 7.

First, we believe that the *replicated* metadata registry with a *centralized synchronization* would perform at its best in a scenario where metadata operations are not so frequent within a task, for instance, a workflow which deals with few, very large files. With tasks taking long enough time to process large files, the agent would have sufficient time to synchronize the registry instances and to provide consistency guarantees that enable easy reasoning on concurrency at application level.

Then, the decentralized strategies are expected to perform at their best with workflows managing a large number of small files. These strategies are of our particular interest, since this kind of workflows occur more frequently in large-scale applications. The *non-replicated approach* is foreseen to target workflows with high degree of parallelism (e.g. following a scatter/gather pattern), where tasks and data are distributed across datacenters. As we mentioned, access to metadata remains linear across sites; thus, we anticipate that the scalability and the throughput of the workflow will be preserved even for increased workloads.

Finally, we envisage that the *locally replicated* will fit better for workflows with a larger proportion of sequential jobs (e.g. with pipeline patterns). Workflow execution engines schedule sequential jobs with tight data dependencies in the same site as to prevent unnecessary data movements [72]. With this approach, we ensure that when two consecutive tasks are scheduled in the same datacenter the metadata are available locally. Even when a task is scheduled in a remote site, it will still be able to access metadata in linear time via the hash value.

Details of the implementation and experimental evaluation of this proposal are provided in Part III of this manuscript.

3.5.2 Distributing Workflows Across Datacenters

In general, workflow management systems are intended to operate in computer clusters or grids, and their design sometimes can be extended to run on a single cloud site. In most of the cases, however, multisite execution is not supported. Deploying workflows to several datacenters involves pondering a number of tradeoffs; we particularly distinguish two: partitioning and scheduling.

Partitioning. Breaking down a workflow is the first step for multisite processing; the workflow's performance relies on a well *balanced* distribution of the jobs. In a common homogeneous computing cluster, the workload can be evenly allocated to the available nodes, however, in a multisite cloud this naïve distribution is not enough. Factors as heterogeneity of the resources, size of the dataset and network throughput come into play, and so "balancing" the workload might depend on what we expect to optimize.

One approach is to *minimize inter-site data transfer*. Because of the low throughput on inter-site networks, transferring large files will slow down the execution and should be avoided. This solution will attempt to co-locate jobs where the estimated volume of data in the dependency is large. Another option is to balance jobs according to a *site's computing capacity*, which applies to heterogeneous environments, i.e. where there are computing nodes with different characteristics. in this partitioning, jobs with heavier workload are assigned to datacenters with more computing power. In case all jobs have similar behavior, then a larger number of jobs is allocated to the more powerful resources. These and other partitioning techniques are further discussed in [73].

Scheduling. Workflow management systems running on single-site clouds apply dynamic load balancing algorithms to optimize metrics like makespan, reliability or financial cost [102]. However, in a multisite environment the scheduler needs to manage and synchronize several datacenters.

An efficient way to address multisite scheduling consists of a 2-tier scheduler that exploits two degrees of granularity: job and task. In the first inter-site tier, a global scheduler is in charge of watching over the whole workflow execution; it distributes *jobs* to the available datacenters according to one of the mentioned criteria (minimize data transfer, balance computing power). The second tier works at intra-site level, a local scheduler assigns *tasks* of a job to the executing nodes of the datacenter, and reports its activity and status to the global scheduler [72].

Within the same research project as the present work, a more in-depth work on multisite workflow scheduling has been realized in parallel by Liu [71]. While we have benefited from Liu's contributions during our collaboration, this thesis does not tackle specific problems of multisite scheduling.

3.6 Conclusion

In multisite workflows, the file information is not limited to its name, size, and path to location; it also includes its physical location (machine/site), a list of jobs that link to it and a history of modifications and reads. Altogether, metadata are no longer of a negligible size. This fact dragged our focus to the overloaded and often disregarded metadata handlers, and brought our efforts to relieve such burden with straightforward solutions. However deceptively simple these design principles and strategies might seem, they have proven effective in other big data contexts: Kademlia DHT [76] is a key component in geo-distributed systems like BitTorrent [140]; whereas dedicated distributed caching such as Redis [123] or Alluxio [68] (formerly Tachyon) are nowadays widely used in industry and science [124, 3]. Our goal in this contribution is to bring together these well-known concepts in a smart and adaptive way to a previously unexplored area: multisite workflow metadata management.

Chapter 4

One Step Further: Managing Hot Metadata

Contents

4.1 What is "Hot" Metadata? 46
4.2 Design Principles 46
4.3 Architecture 48
4.4 Discussion 51
4.5 Conclusion 52

In earlier chapters we have stressed the importance of proper metadata handling in multi-site, large-scale applications, and we proposed a set of metadata management strategies that cover the most common workflow dependency patterns. The goal was clear: to reduce the congestion typically present in centralized metadata servers by distributing the work of metadata coordination across multiple sites. Also, by using the right approach for each type of workflow, we anticipate a decrease in the execution makespan, generally affected by the high-latency inter-site communication.

As a next step we want to build on top of our previous ideas and not only distribute the metadata managing effort, but also give priority to relevant metadata in order to process it earlier. To this end, our focus is to explore the metadata access frequency, and identify fractions of metadata that do not require multiple updates. The goal is to enable a more efficient decentralized metadata management, reducing the number of (particularly inter-site) metadata operations by favoring the operations on frequently accessed metadata, which we call *hot metadata*.

4.1 What is "Hot" Metadata?

> ❝ *We define* hot metadata *as the metadata that are frequently accessed during the execution of a workflow, which should be promptly available to the metadata server(s) of the system.* ❞

The term *hot data* refers to data that need to be frequently accessed. They are usually critical for the application and must be placed in a fast and easy-to-query storage [54]. A typical approach is to keep these data in main memory while moving infrequently accessed (*cold*) records to a secondary storage, like flash [66]. In an analogous way, we transfer this concept to the context of workflow metadata management, and we define *hot metadata* as the metadata that are frequently accessed during the execution of a workflow, which should be promptly available to the metadata server(s) of the system. Conversely, less frequently accessed metadata will be denoted *cold metadata* and will be given a lower priority over the network as we explain later in this chapter. The term "hot metadata" has been previously used with similar semantics for flash file systems [109] or distributed caching for grid systems [126], but it had not been applied to scientific workflows. In a multisite workflow execution environment, we distinguish two types of metadata as *hot*: *task* and *file* metadata.

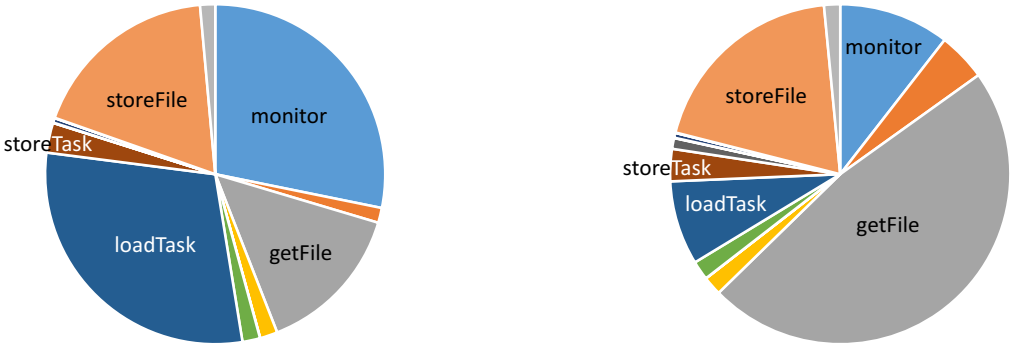
Task Metadata are the metadata corresponding to the execution of tasks, these metadata include the specific execution command for each task and its possible arguments, start time, end time, current status and execution location (site/node). Task hot metadata enable the SWfMS to search and generate executable tasks. During the execution, the status and site of the tasks are queried often in order to search for new tasks ready to execute and to determine if a job is finished. Accordingly, a task's status has to be updated several times along the execution. Therefore, it is important to propagate these metadata quickly to each site.

File Metadata that we consider as "hot" for a workflow execution are those relative to the size, location and possible replicas of a given piece of data, which can be a file, or a block of a file, depending on the data size and the workflow engine's data partitioning mechanism. Knowledge of file hot metadata allows the SWfMS to place the data close to the corresponding execution task, or vice-versa. This is especially relevant in multisite settings: timely availability of file metadata would permit to move data before it is needed, hence reducing the impact of low-speed inter-site networks. In general, other metadata such as file ownership or permissions are not critical for the execution and thus regarded as cold metadata.

The next section describes how the concept of *hot metadata* translates into architectural design choices for efficient multisite workflow processing.

4.2 Design Principles

Several key choices set up the foundation of our architecture. The first consideration, that spans over our different contributions, is the fact that, in a multisite cloud, aiming for a system with a fully consistent state in all of its components at a given moment will strongly



(a) Centralized. (b) Distributed.

Figure 4.1 – Relative frequency of metadata operations in Montage.

compromise the performance of the application. Therefore, we retain the principle of *eventual consistency* in hot metadata management. Two additional decisions complement our principles: *separate processing* and *adaptive storage* for hot and cold metadata.

Separate Management of Hot and Cold Metadata

We studied sample executions of the Montage workflow (described in section 7.1), running 820 jobs and 57 000 metadata operations. They reveal that in a centralized execution, 32.6 % of them are *file* metadata operations (storeFile, getFile) and 32.4 % are *task* metadata operations (loadTask, storeTask), as shown in Figure 4.1a. In contrast, in a distributed run, up to 67 % are file operations, and task operations represent 11 % (Figure 4.1b). These simple runs make evident that a significant percentage correspond to metadata that will not be needed right away or that is used for statistical purposes (mostly monitoring and node/site related operations); yet, in modern workflow execution engines all metadata are handled in the same way.

Following our characterization of *hot metadata* from Section 4.1, we require a model that ensures that:

a) hot metadata operations are managed with high priority over the network, and
b) cold metadata updates are propagated only during periods of low network congestion.

For this purpose, the metadata servers should include a component which discriminates operations as cold or hot *before* propagating them through the network. Our proposal to incorporate such metadata *filter* in the architectures of workflow management systems is presented in Section 4.3.

Adaptive Storage for Hot Metadata

Job dependencies in a workflow form common structures, e.g. pipeline, data distribution and data aggregation [17]. SWfMS usually take into account these dependencies to schedule the job execution in a convenient way to minimize data movements (e.g. job co-location). Consequently, different workflows will yield different scheduling patterns. In order to take

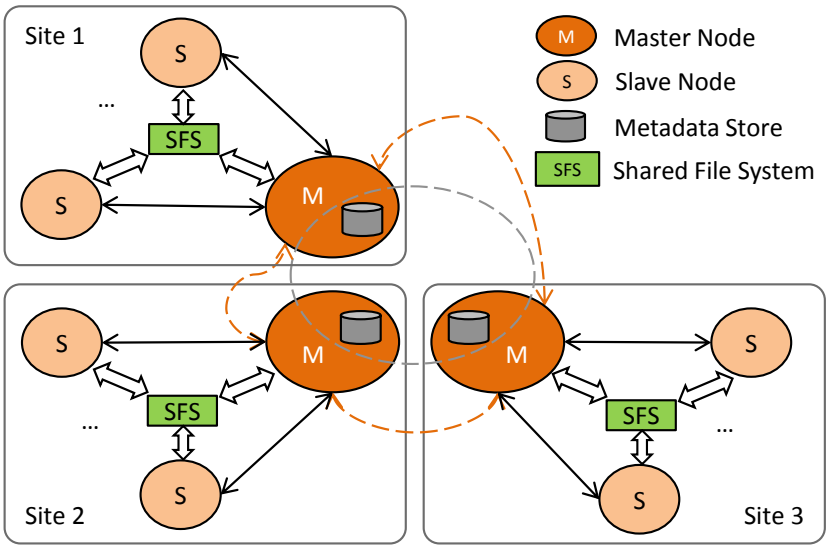


Figure 4.2 – Multisite workflow execution architecture with decentralized metadata management. Dotted lines represent inter-site interactions.

advantage of these scheduling optimizations, we must also dynamically adapt the workflow's metadata storage scheme. However, maintaining an updated version of all metadata across a multisite environment consumes a significant amount of communication time, incurring also monetary costs. In order to reduce this impact, we will apply adaptive storage strategies to our hot metadata handlers during the workflow's execution, while keeping cold metadata stored locally and synchronizing such cold metadata only during the execution of the job. These strategies are based on those presented in Chapter 3. In the next section we recall our decentralized adaptive strategies and explain how hot metadata management is tailored in each of them.

4.3 Architecture

Previously we proposed hybrid approaches combining decentralized metadata and replication to optimize large-scale multisite workflow execution. In this section, we elaborate on top of such strategies into three fundamental lines:

1. We present an architecture for multisite cloud workflow processing which features decentralized metadata management.

2. We enrich this architecture with a component specifically dedicated to the management of *hot metadata* across multiple sites.

3. We explain how hot metadata awareness can be adapted to our hybrid strategies.

A description of protocols for hot metadata *read* and *write* completes the section. Our model handles both file and task hot metadata. The specifics of the implementation of our architecture coupled with a multisite workflow execution engine, as well as a experimental evaluation of its efficiency, are provided in Part III of this manuscript.

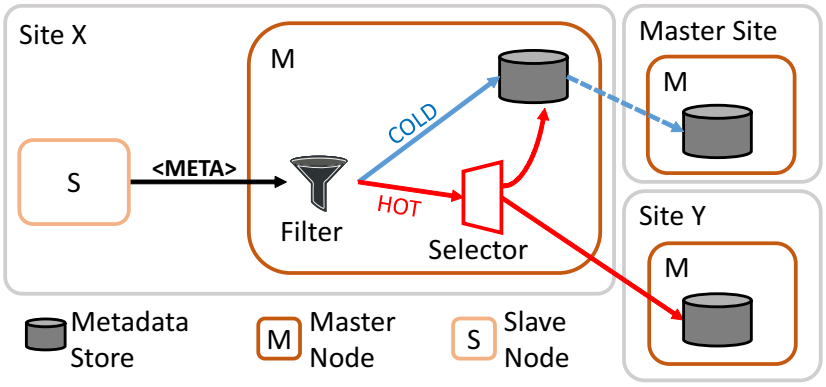


Figure 4.3 – The hot metadata filtering component.

4.3.1 Two-level Multisite Architecture

The basis for our workflow engine is a 2-level multisite architecture, depicted in Figure 4.2. Each level follows a master/slave model, as described below.

1. At the inter-site level, all communication, data transfer and synchronization is handled through a set of master nodes (M), one per site. One site acts as a global coordinator (master site) and is in charge of scheduling jobs/tasks to each site. Every master node holds a *metadata store* which is part of the global metadata storage (shown in a grey dotted circle) that implements one of our distributed strategies and is directly accessible to all other master nodes.

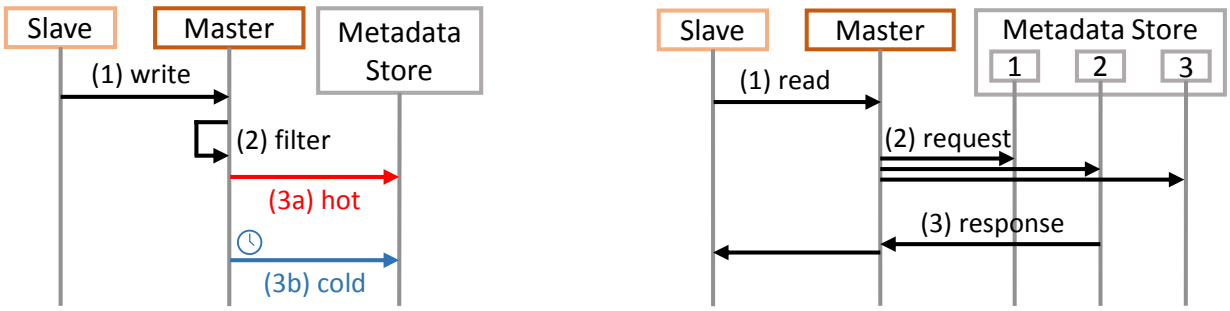
2. At the intra-site level, our system preserves the typical master/slave scheme widely-used today on single-site workflow management systems: the master node schedules and coordinates a group of slave nodes which execute the workflow tasks. All nodes within a site are connected to a shared file system to access data resources. At this level, all *metadata updates* are propagated to other sites through the master node, which classifies hot and cold metadata as we explain next.

4.3.2 Filter for Hot and Cold Metadata

Cold metadata operations must be identified *before* they are propagated to other sites through potentially slower networks. Therefore, we propose to add a filtering component, located in the master node of each site (Figure 4.3). When a master node receives a new metadata operation from a slave, the filter separates hot from cold metadata according to the criteria defined before, favoring the propagation of hot metadata and thus alleviates congestion during metadata-intensive periods. The cold metadata are kept locally in the meantime and transferred later to the master site (dotted line), which holds monitoring and statistical metadata. The storage location of the hot metadata is selected based on one metadata management strategy, as we develop in the coming section.

4.3.3 Decentralized Hot Metadata Management Strategies

We consider the three alternatives for decentralized metadata management explored in the previous chapter. In the following lines, we address their application to *hot* metadata. As



(a) Write.

(b) Read.

Figure 4.4 – Metadata Protocols.

exposed in Figure 3.2, all three scenarios include a metadata server in each of the datacenters where execution nodes are deployed. Unlike the former design, where the metadata registries where isolated entities, in this new 2-level architecture such metadata stores are now located in the master node of each site (Figure 4.2). The strategies differ in the way hot metadata are stored and replicated. We shortly recall their specificities below and explain how hot metadata entries are processed.

Local without replication (LOC). Every new hot metadata entry is stored at the site where it has been created. For read operations, metadata are queried at each site and the site that stores the data will give the response. If no reply is received within a time threshold, the request is resent. This strategy will typically benefit pipeline-like workflow structures, where consecutive tasks are usually co-located at the same site.

Hashed without replication (DHT). Hot metadata are queried and updated following the principle of a distributed hash table (DHT). The site location of a metadata entry will be determined by a simple hash function applied to its key attribute, *file-name* in case of file metadata, and *task-id* for task metadata. We expect that the impact of inter-site updates will be compensated by the linear complexity of read operations.

Hashed with local replication (REP). We combine the two previous strategies by keeping both a local record of the hot metadata and a hashed copy. Intuitively, this would reduce the number of inter-site reading requests. We expect this hybrid approach to highlight the trade-offs between metadata locality and DHT linear operations.

4.3.4 Multisite Hot Metadata Protocols

The following protocols illustrate our system's *metadata operations*. As we mentioned, metadata operations are triggered by the slave nodes at each site, which are the actual executors of the workflow tasks.

Metadata Write. The process is depicted in Figure 4.4a: a metadata record is passed on from the slave to the master node at each site *(1)*. Upon reception, the master node will filter the record to determine if it corresponds to hot or cold metadata *(2)*. Hot metadata are assigned by the master node to the metadata storage pool at the corresponding site(s)

according to different metadata strategies presented above *(3a)*. On the other hand, cold metadata are kept locally and propagated asynchronously to the coordinator site during the execution of the job *(3b)*.

Metadata Read. Each master node has access to the entire pool of metadata stores so it can retrieve hot metadata from any site. The *read* process (shown in Figure 4.4b) is as follows: First, a slave issues a read request *(1)*. When a read operation takes place in a remote location, a master node sends a request to each metadata store *(2)* and it processes the response that comes first *(3)*, provided such response is not an empty set (which would mean that such store does not keep a copy of the record). This mechanism ensures that the master node gets the required metadata in the shortest time.

4.4 Discussion

4.4.1 Challenges for Hot Metadata Management

There are a number of implications in order to effectively apply the concept of *hot metadata* to real workflow management systems. At this stage of our research, we have applied simple yet efficient solutions to these challenges. In the next lines we present two common questions that derive from the definition of hot metadata.

How to decide which metadata are hot? In this contribution, we have empirically chosen task and file metadata as *hot*, since they have statistically proven to be more frequently accessed by the workflow engine that we use (cf. Section 4.2). However, a particular SWf might actually use other metadata more often. Since workflows are typically defined in structured formats (e.g. XML files), another way to account for user-defined hot metadata would be to add a property to each job definition where the user could specify which metadata they consider as *hot*. Moreover, hot metadata could be identified dynamically by the workflow engine by running training executions of the application, but this remains an open issue. Either way, an interesting next step in the research agenda is to implement an environment allowing for both user-defined and dynamically-identified hot metadata. In the next section we briefly discuss said *dynamicity* of hot metadata.

How to assess that such choice of hot metadata is right? Intuitively, a given selection of hot metadata would render the workflow execution faster; however, in reality this evaluation is not trivial in a multisite environment. Metadata are much smaller than the application's data and handling them over networks with fluctuating throughput may produce inconsistent results in terms of execution time. Nevertheless, an indicator of the improvement brought by an adequate choice of hot metadata, and which is not time-bounded, is the *number of metadata operations performed*. In our experimental evaluation (Chapter 7) we present results in terms of both execution time and number of tasks performing such operations.

4.4.2 Towards Dynamic Hot Metadata

Scientific applications evolve over their execution. This means that at a given point, some data might no longer be as relevant as they were initially; in other words, hot data become

cold, or vice-versa. In the case of hot to cold data, file systems might remove them from the fast-access storage or even delete them; conversely, data that become relevant can be promoted to fast storage. Some frameworks assess the data "temperature" *offline*, i.e. they perform a later analysis on a frequency-of-access log to avoid overhead during the operation [66], however, this method is only useful when there are subsequent runs. More interestingly for us, *online* approaches maintain a rank on the frequency of access to the data alongside the execution, for example in adaptive replacement cache [77]. This phenomenon certainly occurs also at the metadata level; so, how could we handle these "temperature" changes in a multisite workflow execution engine? We look into the two situations.

Promoting Cold to Hot Metadata. User-defined hot metadata as discussed above would not allow metadata to be dynamically promoted, since an XML workflow definition file is rather static. However, we can build on this idea and integrate the described online ranking: given a workflow defined through an XML file (or any other definition language), a list of metadata attributes could be passed to the execution engine in the same file; then, the engine would monitor the access frequency of each of such attributes and periodically produce a ranking to verify whether an attribute is required more often, and thus *promote* it to hot metadata. The maximum number of attributes allowed as *hot* metadata could be also dynamically calculated according to the aggregated size of the metadata stores.

Downgrading Hot to Cold Metadata. Less frequently accessed metadata could also be identified using the same attribute ranking approach as above. Upon identification, degrading some metadata to *cold* would also require a collection mechanism that ensures that metadata previously considered hot are removed from fast-access storage. Moreover, this action should take place during not-congested periods, or at the end of the execution so that it does not incur overhead. Taking one step further, we can consider an algorithm that determines the probability that metadata could become hot again later in the execution based on historical data; such metadata could be left in the storage, preventing I/O congestion.

To avoid interfering with the workflow scheduling and execution processes, these scenarios should ideally be implemented transparently within the metadata storage system. We consider them as a potential extension to our work, but we do not implement them.

4.5 Conclusion

In this chapter we proposed an improvement to metadata handling for geo-distributed workflows, namely distinguish *hot* from *cold* metadata through a filter located in each master node. A characterization of a multisite 2-tier architecture to handle hot metadata complements our contribution. In Chapter 6, we present details on how our hot metadata management strategies are implemented into a workflow execution engine. Then, in Chapter 7, we evaluate their performance using real-world workflows, with a zoom on *multi-task* jobs.

Chapter 5

Evaluating Elasticity Factors in Workflow Applications

Contents

5.1 Dynamic Data Streams: Motivations and Challenges 54
5.2 Towards Smart Elastic Provisioning on Clouds 55
5.3 Use Case: Spatial Data for Urban Dynamics 58
5.4 Discussion: Metadata Role in Data Location-Aware Elasticity 59
5.5 Conclusion 59

In the previous two chapters we presented strategies for metadata-driven optimization of workflows on clouds. The improvements were twofold:

- Decentralized management as opposed to the centralized state-of-the-art solutions.
- Hot metadata identification for a more efficient execution.

These solutions can be validated in workflow execution engines as we will present in the next part of this manuscript (Chapters 6 and 7); also, they use synthetic scientific workflows constructed through formal definition rules (e.g. an XML file). For the sake of completeness, it is of our interest to broaden our scope to real-world workflow applications, which are more loosely defined and are not necessarily executed in a specialized workflow management system. This chapter comments on our experience with one such workflow: a sequence of jobs to process geo-tagged data for social analysis. These data are often sourced from *streams*, whose throughput changes *dynamically* along time. The scientific workflow is currently in use by a community devoted to geospatial innovation.

The work introduced next corresponds to a Summer internship at Argonne National Laboratory and represents a part of a larger project that went on after the internship was over. Our involvement was at an early stage and therefore the results are preliminary observations towards the implementation of the solution described.

5.1 Dynamic Data Streams: Motivations and Challenges

> *❝ Dynamic data streams and their computation requirements are unsteady. ❞*

Streams are continuous sequences of data, generated simultaneously by a large number of sources, and usually in small sizes (order of kilobytes) [11]. Examples of streams are log files in a system, online gaming activity records, or weather sensors' measurements.

The number and size of dynamic data streams produced today by sensing and experimental devices as well as social networks is rapidly growing. This is simply because there are more users and sensors generating such data, or due to an increasing complexity in the data structures, e.g. more accurate environmental measuring. Currently, around 6 000 tweets are produced every second [137], and satellites orbit the Earth generating 3.5 TB of geographic data every month [96]. With the growth in these major data sources, scientists are given an unprecedented opportunity to explore a variety of environmental and social phenomena ranging from understanding of weather and climate to population dynamics; with application in as diverse problems as natural disasters mitigation, unemployment assessment and health care. A large percentage of these data streams correspond to data embedded with geographic location information, denoted *spatial data*.

One of the main issues is that dynamic data streams and their computation requirements are unsteady: sensors or social networks may generate data at highly variable rates, processing time in an application may significantly change from one stage to the next one, or different phenomena may simply generate different levels of interest and thus require more (or less) accurate data. Computing clouds allow us to cope with such dynamicity by allocating computational resources on demand, for short periods of time, at an acceptable cost, and within a controlled environment.

Using clouds to process dynamic data streams is not straightforward, though. An application may yield a different performance depending on the hosting infrastructure, thus we require to pay special attention to how and where to schedule cloud resources for these data. We identify two scenarios of such data dynamism.

- Data fluctuates in size and processing needs:
  - Streams generate data at varying rates.
  - Even within a single computation, the dataset size varies along stages. Intermediate data can be much larger than the original input.
- More users are coming to work with the data:
  - Several users connected to a platform simultaneously to run different computations.
  - The number of concurrent users and their needs are unpredictable.

To handle this variability, there are few mechanisms for automatic elastic provisioning in public clouds (e.g. Azure Autoscale [92], Amazon Auto Scaling [12]), yet they use rather *naïve* scaling techniques in the sense that all the virtual machines are created from the same image. We aim for a *smart scaling* approach that enables us to choose the source image from a catalog, selecting the right resource according to the application's current performance and computing needs.

In several cases, these data are not only unpredictably generated, but also produced and stored in different geographic locations, for instance, the Earthscope project for seismology and geophysics accounts for 1200 stations all over the USA, producing about 1 TB of data every six weeks [46]. Therefore, it is not enough to simply deploy a new virtual machine as a response to bursts in data generation (elastically), we also need to provision them close to the data. The aforementioned auto-scaling cloud services do not take data location into account.

To summarize, several challenges arise with location-aware, smart elastic provisioning.

1. Under which conditions is the above *naïve* scheduling approach enough (i.e. schedule a new resource from the same image and wherever there is available space)?
2. Can we model an application's performance in a given infrastructure? Which parameters are relevant for this purpose?
3. What is the impact of data locality? With multiple data sources and large datasets, what is the tradeoff to schedule close to the data?

As explained in the next section, our contribution to this project has a focus on the second challenge, with the objective to identify key parameters that enable to model the performance of distributed workflow applications. Later, in Section 5.3, we present the use case workflow that backed up this work.

5.2 Towards Smart Elastic Provisioning on Clouds

There are two stages of the project in which we got involved: First, we devised and executed experiments to identify scalability triggers in a real-world workflow, which can be modelled for *smart* scaling. Then, we participated in the conception of an architecture for an elastic provisioner that incorporates models and policies to select the right image from a catalog. It is important to mention that the implementation of the provisioner was out of our scope due to time constraints; however, in Section 5.2.2 we discuss how this would be achieved.

5.2.1 Evaluating Scalability Factors for Smart Appliances

Generally, in a scalable application running on the cloud, the new virtual machines are instantiated from images pre-configured with the required software and settings, called *appliances*. An appliance consists of an image containing the application plus the necessary additional software and environment configurations; every new instance is a copy of the original appliance. Sometimes, in order to host powerful applications, these appliances correspond to costly large machines with terabytes of storage, tens of cores or hundreds of gigabytes in memory. Significant grows in data generation or concurrent requests would well justify the creation of a new large VM; however, a small data burst in the execution might only require resources a little beyond those limits and for a short period. With the current *naïve* approach, a big VM would still be instantiated even for the second case, incurring unnecessary expenses, especially when cloud resources are charged/reserved in a per-hour basis. Also, as we previously mentioned, resource scaling in popular clouds does not take into account data location.

Figure 5.1 – Smart appliances will include a performance model.

To avoid the above situation, the project's aim is to produce *smart* appliances. As shown in Figure 5.1, a *smart* appliance will incorporate a module where the performance of the application could be *modelled*. This module would predict the optimal size and configuration for an appliance at the moment of provisioning. The goal is to generate a *statistical model* of the application, based on historical data; to this end, several steps are required:

1. Find parameters which affect the application's performance.
2. Collect a representative training data set for several scenarios.
3. Run the application with different values for the parameters of interest.
4. Identify a suitable basis function to model.

We tackled the first three steps of this process through the development and execution of an experiment plan. We analyzed different stages of our use case application (described in Section 5.3) in terms of data uploading time, data replication, execution time and parallelism. The realization of this plan allowed us to present details into tuning these configuration parameters to extract the best performance, identifying under-utilized resources, tradeoffs for data partitioning and replication, and points of resource saturation.

An in-depth report on the setup and results of these experiments, carried out on two sites of the Chameleon cloud [31], is given in Section 7.4. A plausible next step would be to integrate smart appliances into a full system for elastic provisioning. Our second contribution was a proposal for such a system, which we portray below.

5.2.2 Designing a Smart Elastic Provisioner

Jointly with a team of expert application users, we devised an architecture for smart elastic cloud provisioning. We took into account the two mentioned factors that generate variability in resource usage: data arrival bursts and concurrent user requests. The proposed architecture is depicted in Figure 5.2 and is composed by the following elements.

Data Storage. Data streams produced by sensors, scanners or mobile devices arrive at different rates to the cloud system. Data can be handled by different storage services, to match the size of the data and the application's specific needs. For example, in-memory relational [108] or NoSQL [123] storage can be used for immediate data access.

Portal. Users of different applications submit requests through a portal, which can be a web or a command line interface. The number of concurrent users/requests is unpredictable; therefore, the portal should be able to scale accordingly, to guarantee that all requests are cared for. Such requests are forwarded to the provisioner for scheduling and resource allocation.

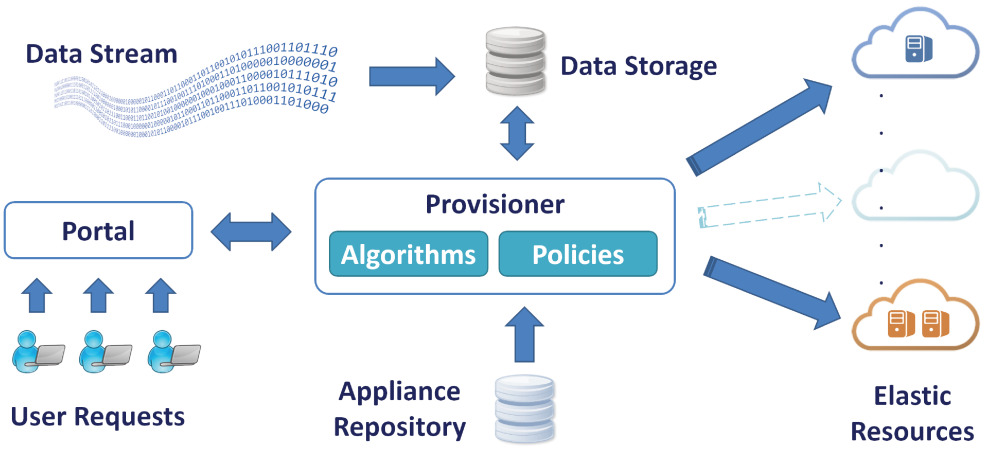


Figure 5.2 – Architecture of the elastic provisioning system.

Provisioner. The provisioner is the core component of the system. Its functionality is supported by the Nimbus Phantom auto-scaling service [65], whose decision engine accounts for user defined *policies* to create/delete virtual machines. New *smart* policies will be added to account for the observations obtained from the experimental evaluation. These policies will define not only *when* to provision, but also *where* (data locality) and *which* appliance (in terms of size and settings). The provisioner uses algorithms applied to the set of policies to take decisions according to the current state of the application(s).

Appliance Repository. A number of customizable images are stored in the repository. A smart appliance will be selected based on the application's requirements. Different configurations in terms of disk size, memory and application specific parameters will be possible.

Elastic Resources. The provisioner deploys and removes computing nodes to different cloud sites. In Phantom it is possible to deploy even to different cloud providers. The choice of site/provider can be motivated by the available types of computer in each site, or by the proximity of the data to process.

How to cope with data bursts and concurrent users? Data streams are processed in a best-effort manner. Stream processing systems are usually able to handle "predictable" fluctuations in data-arrival rate: they add resources during regularly busy periods of the day and reduce them otherwise, this is the case of scaling in games, for example [139]. However, there are unpredictable events that trigger the arrival of unusual data loads which might challenge/reduce the system's responsiveness.

In our proposal, time windows for predictable bursts can be defined to anticipate the creation of resources, in the same way as the previous example. In addition, *smart* policies based on statistical models of the applications will be improved over time, using historical data every time they are run. Atypical data bursts can be addressed based on these *smart* policies. They will determine the right appliance (location, size, configuration) to provision

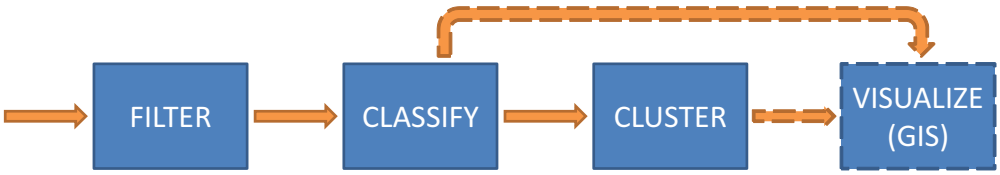


Figure 5.3 – A workflow for spatial data synthesis (analysis of urban dynamics).

according to: (i) where the data is located; (ii) what the predicted data growth is (growth is difficult to predict, but the models could give an approximate); or (iii) what the expected time to completion is, based on the input size.

A similar approach can be taken to deal with bursts of concurrent users. A base policy will determine the expected amount of users at a given time, to foresee the required resources. On the other hand, *smart* policies will help to respond to irregular peaks of concurrent users. The provisioner will base on models of: (i) resources needed per user for a given application; or (ii) computing capacity required by an application with a given size of input data.

5.3 Use Case: Spatial Data for Urban Dynamics

In this project we worked closely with members of the CIGI Laboratory[1] from the University of Illinois at Urbana-Champaign, a group of scientists devoted to generate tools that enable *spatial data* (geo-tagged data) visualization and analysis. They provide a framework, CyberGIS [36], for integrating interoperable and reusable software, linking large cyberinfrastructures to achieve scalability with respect to the size of the problems, the amount infrastructure resources, and the number of concurrent users.

Specifically, our use case was in the context of the DIBBs project, focused on *spatial data synthesis for understanding urban dynamics*. The application evaluates data streams obtained through the Twitter API, processing millions of records representing days of geo-tagged tweets. The data is used to estimate individuals' home and work relocation, by identifying users' trajectories and top visited locations, to then correlate them with unemployment rates in a delimited region. The application consists of four phases, illustrated in Figure 5.3 and defined as a workflow by the DIBBs team, as follows.

Filtering. First, the duplicated, retweeted or corrupted tweets are identified and discarded. Then, irrelevant or sensitive information such as tweet text and URLs is filtered out.

Classification. The dataset is narrowed down to those tweets created within a specified area (e.g. a neighborhood, a city or a state), by looking into their geographic coordinates. Once the relevant tweets have been selected, tweets are classified by unique users.

Clustering. A clustering algorithm is run for every user, it groups tweets generated at neighboring locations, within meters from each other. The objective is to pinpoint the top frequented locations.

[1] CyberGIS Center for Advanced Digital and Spatial Studies. CyberInfrastructure and Geospatial Information Laboratory, UIUC. http://www.cigi.illinois.edu/.

Visualization. Using GIS and maps APIs, the clustered tweets can be displayed in a virtual map to visually identify the corresponding location or place of interest. It can take data both from the classification and the clustering phases. This stage had not been fully developed by the time this work was carried out.

Since the last two phases were not yet available, we leveraged releases of the *filtering* and *classification* jobs for our experimental evaluation (Section 7.4).

5.4 Discussion: Metadata Role in Data Location-Aware Elasticity

In the formulation of our problem we asked ourselves *what is the impact of data locality* in smart scaling, basing on the case where data are produced/stored in several geographic locations. Ideally, a new resource must be created as close as possible to the data that it will use. To make decisions on elastic provisioning in this data-sensitive way we need information about the data, in particular, data location and data size metadata.

In previous chapters we tackled the problem of multisite metadata handling, specifically for geo-distributed cloud applications, through a set of metadata registries that implement hybrid storage strategies. A similar approach could be embedded to the smart scaling architecture, metadata can be tracked and provided to the resource provisioner as an input for allocation decisions, using distributed metadata handlers.

- An instance of the metadata registry should be included in each cloud site in the system, such instance will keep track of the available data in that site and will update the provisioner continuously.
- Data-driven policies should be added to the *Policies* module, they will serve the provisioner to decide on the most data-wise convenient site to deploy a new resource.

The improvement brought by implementing this approach could be tested in three scenarios using a geo-distributed application.

- Naïve resource provisioning without data location awareness.
- Location-aware with a centralized metadata server.
- Location-aware with distributed metadata server.

This notion of embedding multisite metadata handlers for smart scaling has been discussed with our hosts at Argonne National Laboratory. It is out of the scope of this contribution due to time constraints, but will be considered for the implementation of the smart elastic provisioner.

5.5 Conclusion

We presented an experience with a real-world workflow on the cloud. This collaboration helped us learn actual challenges of deploying such applications to cloud infrastructures. Beyond improving the overall execution time, which was our target in earlier chapters, in this project we faced scalability and, implicitly, fault tolerance issues. Timely provisioning

of cloud resources improves the workflow's performance and in some cases, as we will see in our experimental evaluation (Section 7.4), prevents it from crashing. Smart elastic cloud scaling will allow an application to respond to arbitrary changes in the processing/storage needs; but more importantly it will help to optimize the cloud resources utilization while maintaining the expected performance and service level.

This chapter concludes the list of contributions brought by this thesis. The remaining of the manuscript details the implementation and evaluation of these ideas.

Part III

Implementation and Results

Chapter 6

Implementation Details

Contents

6.1 Distributed Metadata Registry 63
6.2 DMM-Chiron 65
6.3 Conclusion 68

AFTER presenting our three contributions, this chapter represents the preamble to the validation of our proposals. Here, we introduce the implementation aspects of the two architectures proposed in this thesis.

The first section explains how workflow metadata is distributed to multiple decentralized registries. We describe how each of these registries can communicate with the entire set of execution nodes and how the metadata is stored and retrieved from them. The second section details how we incorporated distributed metadata handling to a multisite scientific workflow management system. The appended metadata management module implements the *hot* metadata protocols and interacts with several components of the system, providing efficient access to *hot* metadata and controlling the delayed propagation of *cold* metadata.

6.1 Distributed Metadata Registry

Our metadata management strategies, proposed in Chapter 3, are designed as a general purpose multisite metadata handling paradigm and not aimed to a specific cloud platform. The execution nodes can be mapped to regular virtual machines, and the metadata registries require in memory key/value store, which can be a generic, open solution such as Redis [123]. For validation purposes, in this implementation we use the Microsoft Azure Cloud [84] as a concrete example to demonstrate how to implement them in practice at Platform-as-a-Service (PaaS) level. We rely on the Azure SDK v2.5 for .NET which provides the necessary libraries for accessing and manipulating Azure features. The architectural overview of the metadata middleware is shown in Figure 6.1 and its components are discussed below.

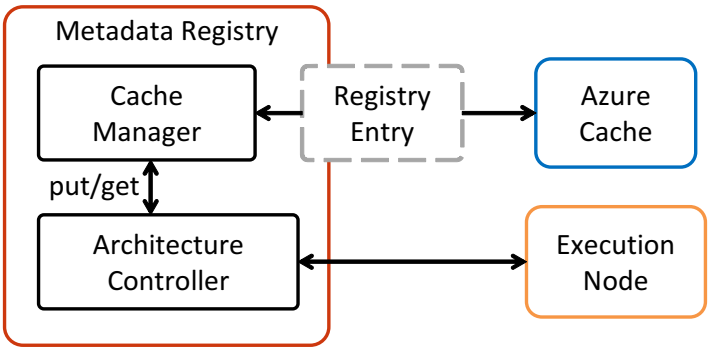


Figure 6.1 – Metadata Registry modules and interaction.

The Metadata Registry stays at the core of our implementation, as it serves as communication channel and distributed synchronization manager between all nodes in the network. From previous experimental evaluations on Azure, we know that in-memory storage access outperforms regular table or BLOB storage by a factor of 10 [135], thus we opted to implement the registry on top of the Azure Managed Cache[1] service [93]. Azure Cache provides a secure dedicated *key/value* cache that can be accessed remotely by means of a URI. Azure Cache allows to store any serializable object in the *value* field of an item. In order to guarantee the durability of our records, least-recently-used eviction and object expiration time properties were disabled. To allow concurrent access to the registry we chose the *optimistic concurrency* model of Azure Cache, which does not pose locks on the registry object during a metadata operation, leveraging the workflow's characteristic that data are written only once. The metadata registry has been implemented in C# and is composed by three elements.

The Cache Manager is an interface with Azure Cache by means of the external .NET library `WindowsAzure.Caching`. It exposes a set of internal *put* and *get* methods to carry out (cache-managed) metadata operations. The manager creates a connection to a cache service specified by a URI; several URIs can be configured in a pool of *cache clients* as shown in Listing 6.1. Using multiple cache clients gives the possibility to create connections to different caches from a single manager, what is particularly needed for the metadata replication. The cache manager is an independent module with simple *put/get* interfaces, this allows to explore different cache alternatives without affecting the application, for instance, Azure Managed Cache could be replaced with Redis Cache [123].

The Registry Entry is the fundamental metadata storage unit. The Cache Manager is able to *put* (write) and *get* (read) Registry Entries from the Azure Cache registry. Every entry is a *key/value* pair and can contain any piece of metadata provided it is serializable and has a unique identifier; in that way a registry can store heterogeneous entries. For our implementation we took the base case of a *file*, where the *key* identifier is the file's name and the *value* field is a set of file locations within the network (site, node). The scope of the registry can be easily extended by defining additional constructors for other serializable custom objects.

[1] By the time this manuscript was produced, Azure Managed Cache has been replaced by Azure Redis Cache as the recommended Azure Cache service.

```
<dataCacheClients>
  <!-- West Europe cache client-->
  <dataCacheClient name="weu">
    <autoDiscover isEnabled="true" identifier="luiscacheweu.cache.windows.net" />
  </dataCacheClient>
  <!-- Central US cache client -->
  <dataCacheClient name="cus">
    <autoDiscover isEnabled="true" identifier="luiscachecus.cache.windows.net" />
  </dataCacheClient>
</dataCacheClients>
```

Listing 6.1 – Defining multiple cache clients.

The Architecture Controller component allows to switch between metadata management strategies. The desired strategy is provided as an input parameter and thus can be dynamically modified as new jobs are executed. According to the selected approach, the controller will request the cache manager to create as many connections as needed. The modularity of our design allows us to add or remove strategies on the fly, using a simple plug-and-play approach, without altering the application flow.

The Execution Nodes are Virtual Machines running on the Azure cloud. We designed three types of instances based on Azure's PaaS-level abstractions Web Roles and Worker Roles, which are grouped in a Cloud Service.

- *The Worker Nodes* execute the application tasks, they create put/get requests for the metadata registry. They are implemented as background engines on top of Azure Worker Roles.
- *The Control Node* manages the application execution. Implemented as an Azure Web Role, it includes a simple web interface to enter configuration parameters. It launches the execution of the application by sending control messages to all the Worker Nodes. Control messages are sent using Azure Queues.
- *The Synchronization Agent* is a Worker Node in charge of synchronizing the metadata registry instances in the *replicated* strategy. It sequentially queries the instances for updates and propagates them to the rest of the set.

6.2 DMM-Chiron

In order to validate our two-layer architecture presented in Chapter 4, we have developed a prototype multisite SWfMS that implements hot metadata handling. It provides support for *decentralized* metadata management, with a distinction between hot and cold metadata. We denote our prototype Decentralized-Metadata Multisite Chiron (DMM-Chiron). Its operational architecture is shown in Figure 6.2 and is described next.

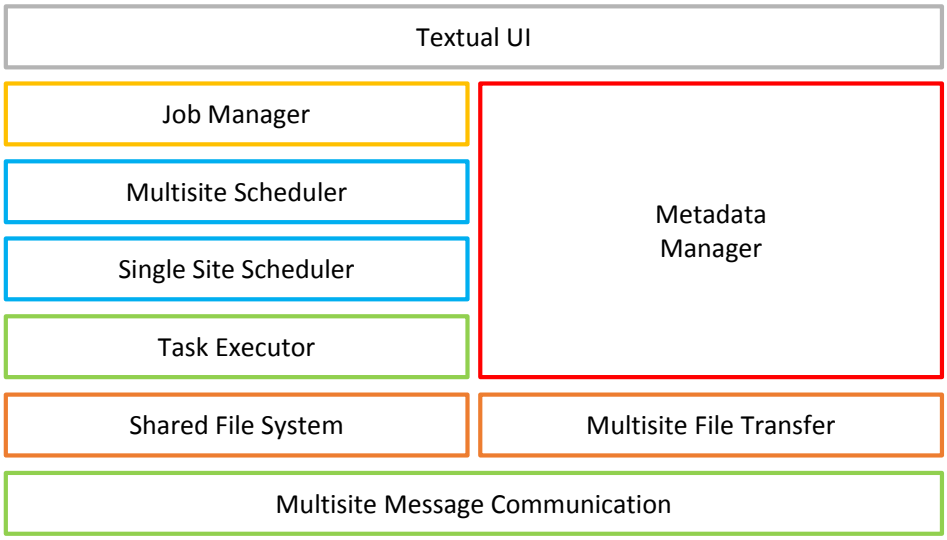


Figure 6.2 – Layered architecture of DMM-Chiron.

6.2.1 Baseline: Multisite Chiron

This work builds on Multisite Chiron [72], a recently released workflow management system specifically designed for multisite clouds. Multisite Chiron is implemented in Java, but it can execute any Unix-compatible application. A full description of this approach, including its focus on multisite scheduling, is provided in [72]. We present a list of its components.

Textual UI. Multisite Chiron exploits a textual UI to interact with users, the workflow configuration parameters and the workflow definition are passed as XML files.

Job Manager. The job manager analyzes the submitted scientific workflow in order to identify unexecuted jobs, for which the input data is ready. The same module generates the executable task as an execution plan on the provenance database.

Multisite Task Scheduler. It is located at the coordinator site. Its function is to distribute the tasks to the available sites. At the moment, the multisite scheduler does not support data-location awareness; therefore, the scheduling considers other load balancing techniques as explained in [72].

Single-Site Task Scheduler. Following a 2-layer hierarchy, a second scheduler distributes tasks to the computing nodes, this is done in a round-robin fashion, assuming that all nodes of the subnetwork have equal computing capacity and have access to the shared file system.

Task Executor. They are the actual computing nodes in charge of running the assigned tasks. Every update to the workflow's metadata originates from them.

Metadata Manager. Along the execution, metadata is handled by the metadata manager at the master site. Since the metadata structure is well defined, we use a relational database, namely PostgreSQL [115], to store it.

Shared File System. All data (input, intermediate and output) are stored in a shared file system at each site. For this implementation we chose to install NFS [122] in every master node, which is mounted as local in each of the slaves.

Multisite File Transfer. Is in charge of file transfers between two different sites. As we previously mentioned, this is done through the public Internet, exploiting a peer-to-peer model for data transfers.

Multisite Message Communication. Exists at the master node of each site. It ensures the synchronization with other sites and the master site. For this implementation, control messages are transmitted leveraging Azure Bus [79], a lightweight service accessible from any location through an URI or API.

To implement and evaluate our approach to decentralized hot metadata management. DMM-Chiron incorporates multisite metadata protocols and a filter for hot metadata. We mainly affect two modules of the system: the *Metadata Manager* and the *Job Manager*.

6.2.2 Metadata Manager

The metadata management service, as proposed in the architecture of our system (Figure 4.2), is a component present in every site where the SWfMS is deployed. It is physically located within the master node of each site. The metadata manager includes a share of the metadata store and the logic to handle it. This logic is composed by two hot metadata elements, namely protocols and filter, that we briefly recall.

The hot metadata protocols were defined in Section 4.3.4. They establish the actions to take upon the reception of a metadata update from a slave node. After classifying the update as hot or cold, the master node decides whether to propagate it immediately to the whole network (hot) or reserve it for later propagation (cold).

The hot metadata filter consists of a set of policies to determine if a given piece of metadata (and its corresponding updates) is hot or cold. For this implementation, the policies are static rules according to historical data of our applications. However, this component can be enhanced with different, dynamic rules or algorithms that determine the "temperature" of metadata, as discussed in Section 4.4.

The metadata manager provides the necessary metadata values for the job manager to generate an execution plan, both file and task hot metadata are required for this process.

6.2.3 Job Manager

The Job Manager is the process that generates the execution plan in the provenance database. The execution plan is the list of executable tasks per job, including their corresponding execution command, parameters and the location of the input data (*file hot metadata*). The job manager also tracks the execution state of a job. If the execution of a job is finished, its dependent jobs may be launched. For a job to be finished it is necessary that all its tasks are marked as finished. A task goes through several stages: from *created*, to *ready* for execution,

to *running*, and finally it will be *finished*. The SWfMS requires to monitor and update the status of each task in near-real time (*task hot metadata*), and the number of tasks can easily reach tens of thousands. When tasks a running in remote locations, there is an expected delay for these updates to be reflected across the multiple datacenters of the system. This would impact the performance and execution time of the workflow: the job manager might not be aware that a job is finished because the execution information is not updated quickly.

From the previous description we can clearly identify the impact of hot metadata. A quick file metadata propagation allows the job manager to know in advance the location of a file and prepare the execution plan. If the file is located in a remote site, the command and parameters to access it are different to a local one. Job-related hot metadata, on the other hand, allow to control the execution flow of the application, specifically by determining the status of each task.

Originally, the verification for job completion was done on the metadata stored at the coordinator site. In DMM-Chiron we implement an optimization according to one of the hot metadata management strategies (Section 4.3.3): for LOC, the local DMM-Chiron instance verifies only the tasks scheduled at that site and the coordinator site confirms that the execution of a job is finished when all the sites finish their corresponding tasks. For DHT and REP, it is the master DMM-Chiron instance of the coordinator site which checks each task of the job across the different sites.

6.3 Conclusion

In this section we have presented implementation details of the two middleware prototypes used in this thesis. Our solutions are based in well-know computing concepts and tools. Nevertheless, we have tailored them to support multisite execution of scientific workflows through efficient metadata management. We used in-memory caching services for quick access of workflow metadata. The location of these metadata was determined through distributed hash tables to reduce search time. As other multisite systems, our metadata registry relies on an eventual consistency model. Hot metadata is a concept that has been used for caching and file systems in grids.

Overall, we have orchestrated these tools and notions in service of workflow management systems, to bridge the gap between single- and multisite execution. The next chapter introduces a series of experiments where our prototypes are evaluated across different scenarios.

Chapter 7

Evaluation

Contents

7.1 Experimental Setup 69
7.2 Decentralized Metadata Management on the Cloud 72
7.3 Separate Handling of Hot and Cold Metadata 78
7.4 Identifying Thresholds for Elastic Scaling 83
7.5 Conclusion 88

THIS section assembles an extensive evaluation of the performance of our decentralized metadata management strategies. We first review the infrastructure and use cases adopted for the experiments. Then, in Section 7.2 we look into the gains brought by handling workflow metadata using our decentralized multisite approach, as opposed to a typical centralized management (proposed in Chapter 3). Later, in Section 7.3, we validate the added benefits of a selectively handling decentralized metadata based on their access frequency (*hot metadata*, detailed in Chapter 4), implemented in a real workflow management engine. Additionally, in Section 7.4, we report on the experiments conducted to identify thresholds for elastic scaling using a real-world workflow, introduced in Chapter 5.

7.1 Experimental Setup

7.1.1 Infrastructure

Two cloud platforms lie at the foundation of this thesis: Microsoft Azure [84] as a *public* cloud, and Chameleon [31] as a *private* one. In section 2.2.2 we gave a general overview of their main features. Now, we present some additional characteristics of them, related to the series of experiments in this chapter. The specific cloud configuration used for each experiment is defined at the beginning of the corresponding section.

Microsoft Azure was our testbed for the majority of experiments. A significantly large amount of resources was made available to us thanks to our partnership with Microsoft Research through the Z-CloudFlow project[1]. Additional resources were obtained from an Azure for Research Award[2]. We recall that Azure's datacenters are distributed across 34 geographic regions, providing infrastructure proximity for clients mostly anywhere on Earth. Our experimental evaluation was conducted using datacenters in Europe and the US. We used small to mid-sized virtual machines, with a maximum of 8 cores per node. Azure Table key/value storage [85] was chosen for logging, since it handles concurrent writes from several nodes in a transparent way. Additionally, we leveraged Azure's PaaS solutions for several purposes, as we will detail later: Azure Managed Cache [93] for the metadata registries, Azure Cloud Services [94] for workflow management, and Azure Bus [79] for control messages.

Chameleon Cloud is the *de facto* infrastructure at the Nimbus team, with whom we collaborated during a summer internship at Argonne National Laboratory [31]. As mentioned previously, Chameleon is a testbed open and enhanced for scientific projects. It includes ~650 nodes (~14,500 cores), 5 PB of storage and is distributed over 2 sites, ANL in the Chicago Area, and TACC[3] in Austin, Texas, connected through a 100 Gbps network. Its infrastructure is composed by 12 standard *cloud units* (48 node racks), nine of them with bare-metal reconfigurability and three with OpenStack KVM, which provides easy access to educational users. Most of the experimentation was carried out on their bare-metal units (called CHI), running CentOS7. On a side note, it is worth to mention that Chameleon went publicly available right at the time of the internship, and several of its current features were under development or did not exist back then. For instance, CentOS was the only available image. Also, part of our experiments helped to test new functionalities like *leases* management.

7.1.2 Use Cases

Two scientific workflows supported our study for the first two contributions of this thesis. Both were defined following Chiron's algebra [100]. We describe them briefly.

Montage is a toolkit created by the NASA/IPAC Infrared Science Archive and used to generate custom mosaics of the sky from a set of images [41]. Additional input for the workflow includes the desired region of the sky, as well as the size of the mosaic in terms of square degrees. We model the Montage SWf using the proposal of Juve et al. [63] as shown in Figure 7.1. The *mProjectPP* job projects single images to a specific scale. *mDiffFit* performs an image difference between a single pair of overlapping images. *mConcatFit* uses least squares to fit a plane to an image. *mBgModel* uses the image-to-image difference parameter table to interactively determine a set of corrections to apply to each image to achieve a "best" global fit. *mBackground* removes the background from a single image. Using the output of both mProjectPP and mBgModel. *mImgTbl* prepares the information for putting the images

[1] Z-CloudFlow - Data Workflows in the Cloud. Microsoft Research - Inria Joint Centre https://www.msr-inria.fr/projects/z-cloudflow-data-workflows-in-the-cloud/.

[2] Microsoft Azure for Research Awards https://www.microsoft.com/en-us/research/academic-program/microsoft-azure-for-research/.

[3] Texas Advanced Computing Center.

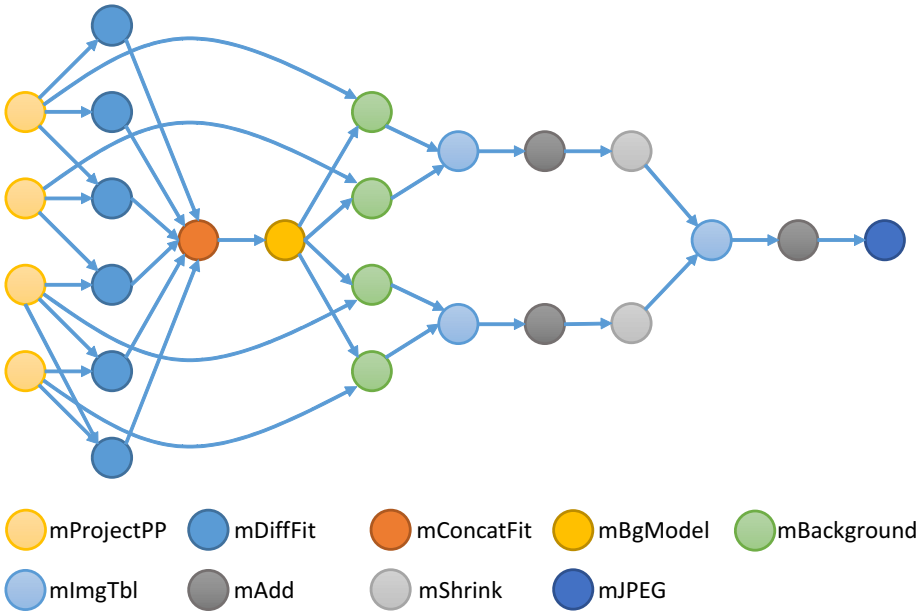


Figure 7.1 – The Montage workflow.

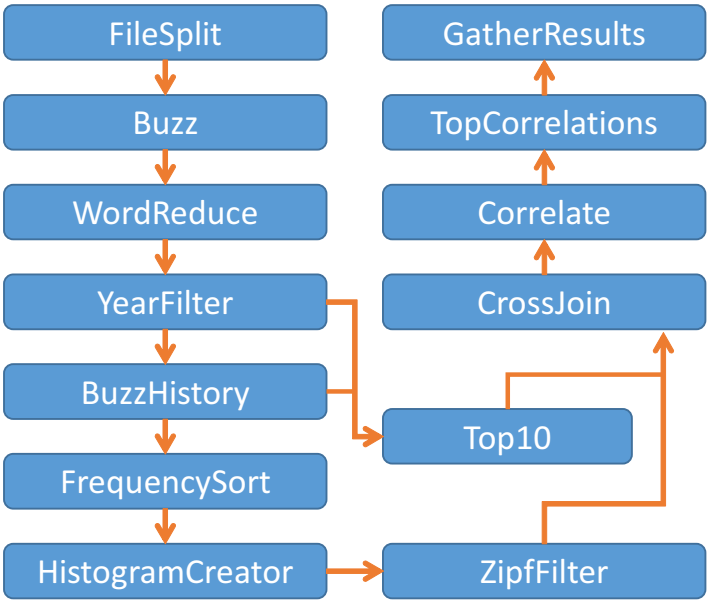


Figure 7.2 – The Buzz workflow.

together. *mAdd* generates an output mosaic and the binning of the mosaic is changed by *mShrink*. Finally, *mJPEG* creates a JPEG image from the mosaic.

BuzzFlow is a data-intensive SWf that searches for trends and measures correlations in scientific publications [43]. It analyzes data collected from bibliography databases such as DBLP or PubMed. *Buzz* is composed of thirteen jobs, as shown in Figure 7.2. *FileSplit* gathers the information from bibliography databases. The *Buzz* job uses the gathered information to identify buzzwords, i.e. a word or phrase that can become popular for a specific period of time. *WordReduce* organizes these publications according to buzzword and publication year and computes occurrences of the buzzwords. *YearFilter* selects buzzwords that appeared in the publications after a specific time. *BuzzHistory* job creates a history for each buzzword. *FrequencySort* computes the frequency of each buzzword. With this information, *Histogram-Creator* generates some histograms with word frequency varying the year. *Top10* selects ten of the most frequent words in recent years. *ZipfFilter* selects terms according to a Zipf curve that is specified by word frequency values. *CrossJoin* job merges results from Top10 and ZipfFilter. *Correlate* computes correlations between the words from Top10 and buzzwords from ZipfFilter. Using these correlations, *TopCorrelations* takes the terms that have a correlation greater than a threshold. Finally, *GatherResults* presents these selected words with the histograms.

The use case corresponding to the contribution presented in Chapter 5 has not been defined under the same algebraic approach, nor has been evaluated under the same criteria, as we will elaborate in Section 7.4. This use case, involving an application for spatial data analysis, was detailed in Section 5.3. Henceforth, we focus on the results of the evaluation of our three contributions.

7.2 Decentralized Metadata Management on the Cloud

This first set of experiments evaluates the strategies presented in Chapter 3. The main goal is to highlight the benefits of implementing distributed metadata management for multisite scientific workflows, as opposed to the usual centralized approach. Therefore, the experiments compare our three strategies: replicated, decentralized non-replicated, and decentralized replicated (Section 3.4), to the centralized solution.

Cloud Configuration

Our testbed consisted of nodes distributed in four Azure datacenters: two in Europe, North (Ireland) and West (Netherlands), and two in the US, South Central (Texas) and East (Virginia). We used up to 128 *Standard_A1* virtual machines, each consisting of 1 core and 1.75 GB of memory. For the Metadata Registry we deployed one *Basic*[4] 512 MB instance of Azure Managed Cache per datacenter. All experiments are repeated at least five times and the reported figures are the average of all runs. To hinder other factors such as caching effects and disk contention, the metadata entries posted to the registry (e.g. create, update or remove) correspond to empty files.

[4]The *Basic* cache was the base offer of Azure Managed Cache, *Premium* offers would reach up to 150 GB.

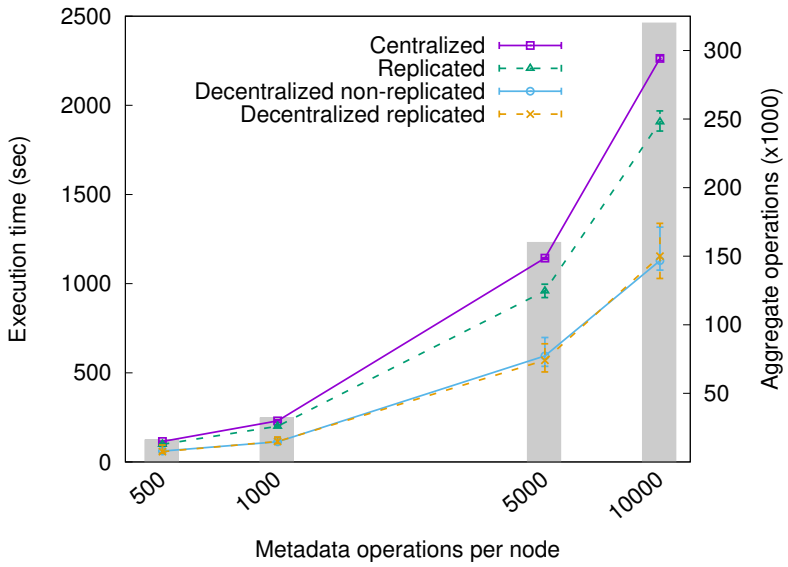


Figure 7.3 – Average execution time for a node performing write metadata operations.

7.2.1 Impact of Metadata Decentralization on Makespan

We claim that the efficiency of our decentralized strategies becomes more evident in large-scale settings. The goal of the first experiment is to compare the performance of our implementation to the baseline centralized data management as the number of files to be processed increases. For this purpose, we keep a constant number of 32 nodes evenly distributed in our datacenters (i.e. 8 nodes per datacenter), while varying the number of entries to be written/read to/from the registry. To simulate concurrent operations on the metadata registry, half of the nodes act as writers and half as readers (i.e. 4 readers and 4 writers per datacenter). Writers post a set of consecutive entries to the registry (e.g. *file1*, *file2*, etc.) whereas readers get a random set of files (e.g. *file13*, *file201*, etc.) from it. We measure the time required for a node to complete its execution, and obtain the average time for completion of all the nodes for each strategy. Figure 7.3 shows the results.

We observe that for a rather small number of processed entries, none of our strategies significantly outperforms the centralized baseline in terms of overall execution time. They represent a gain of slightly more than 1 minute in the best case, which is rather low in our context. We infer that for small settings — up to 500 operations per node — a centralized approach remains an acceptable choice, since the effort of implementing a distributed solution would not be compensated by a meaningful gain. However, as the number of operations grows, the improvement achieved by our strategies becomes more evident. Full metadata replication brings an average gain of 15 %; we attribute this simply to the fact that the metadata management duty is now distributed. In particular, the decentralized strategies (with and without replication) yield up to 50 % time gain compared to the centralized version. In the figure, the grey bars (linked to the right y-axis) indicate the aggregated number of operations in one execution. At the largest scale, a 50 % gain represents 18.5 minutes in a test with 320 000 metadata operations.

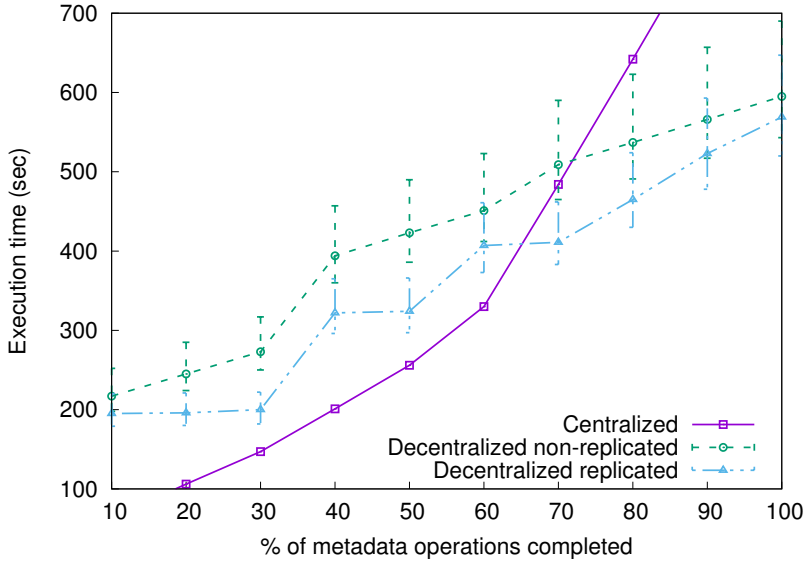


Figure 7.4 – Percentage of operations completed along time by each of the decentralized strategies: non-replicated and with local replication. Workload: 5 000 operations per node.

Decentralized strategies: completion time vs. speedup. An interesting observation is that both decentralized approaches seem to overlap. The time for completion in each strategy depends on the time taken by the last active node to execute its very last operation. The curves in Figure 7.3 do not reflect then the progress of the nodes activity before their completion. For that reason, we do not clearly see an advantage of one decentralized strategy over the other. Therefore, in Figure 7.4 we zoom on the internal execution of the two decentralized strategies (non-replicated and locally replicated), by analyzing their progress towards completion. We also show as reference the average progression of the generic centralized approach.

We notice that during most of the execution, particularly between 20 % and 70 % progress, we get a speedup of at least 1.25 using local replication. This remark is crucial for data provisioning in a distributed scientific workflow: the time gain implies that data location information can be known by the whole network in anticipation, and represents the possibility to move the data between sites before it is needed.

The centralized approach has a fairly good start on average. However, as the execution advances, it slows down in a near-exponential behavior, reaching up to twice the time for completion compared to the decentralized ones. This delay is due to the increasing overload of operations on the centralized metadata registry, doubled by the accumulated latency of distant nodes performing remote operations.

Impact of the geographical location of a datacenter. Finally, we focus on the best and worst performance of the decentralized approaches shown in Figure 7.4. In most of the points plotted beyond 30 % of operations completed, we noticed that the difference between the fastest and slowest executions is in the order of minutes (vertical "error" bars), which is beyond 10 % of the execution time. To understand this significant variation, we made a careful analysis of the results log per site. We noticed a clear impact of the distance between sites on the metadata handling performance. If we consider a site's *centrality* as the average

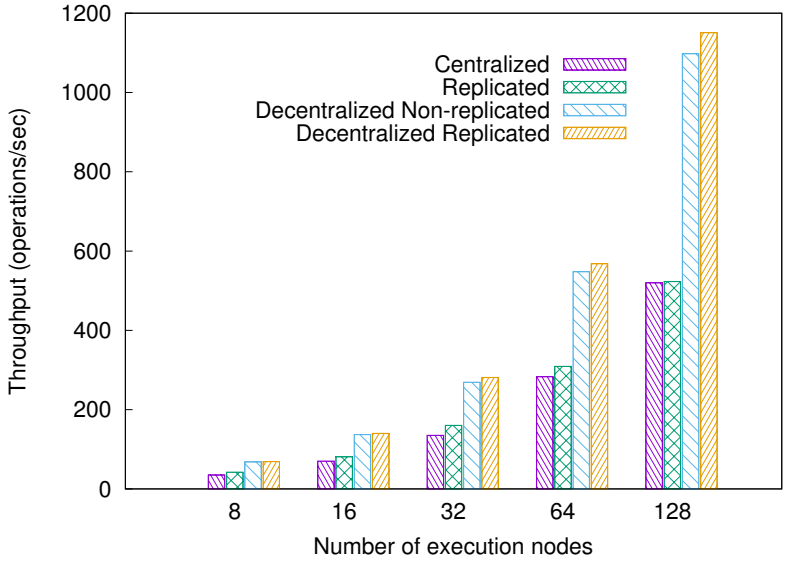


Figure 7.5 – Metadata throughput as the number of nodes grows.

distance[5] from it to the rest of the datacenters, the best performance in both cases corresponds to the nodes executed in the most *central* datacenter, East US. Conversely, the slowest executions came from the least *central* datacenter, South Central US. This observation could be extrapolated to a complete multisite workflow execution. If the coordination component (scheduler, manager) were placed in the most *central* datacenter, control messages would be delivered faster, hence improving the workflow's makespan.

7.2.2 Scalability and Concurrency Sensitivity

In our next experiment, we evaluate the performance of our strategies when the number of nodes increases. Note that since each node acts also as a metadata client, this scaling translates into an increased concurrency as well. First, we measure the metadata throughput when increasing the number of nodes from 8 up to 128, with a constant workload of 5 000 operations per node. In Figure 7.5 we observe that the decentralized implementations clearly win: they yield a linearly growing throughput (given in operations per second), proportional to the number of active nodes. We only notice a performance degradation in the replicated scenario, intensified beyond 32 nodes. We assert that as the number of nodes grows, the single replication agent becomes a performance bottleneck; however, in smaller settings of up to 32 nodes, it still behaves efficiently.

To get a clearer perspective on the concurrency performance, we measured the time taken by each approach to complete a constant number of 32 000 metadata operations. Our results (Figure 7.6) were consistent with the previous experiment, showing a linear time gain for the centralized and decentralized approaches and only a degradation at larger scale for the replicated strategy.

[5]Distance determined in terms of latency between datacenters, which generally generates a ranking similar to physical distance. Latencies obtained from http://www.azurespeed.com/.

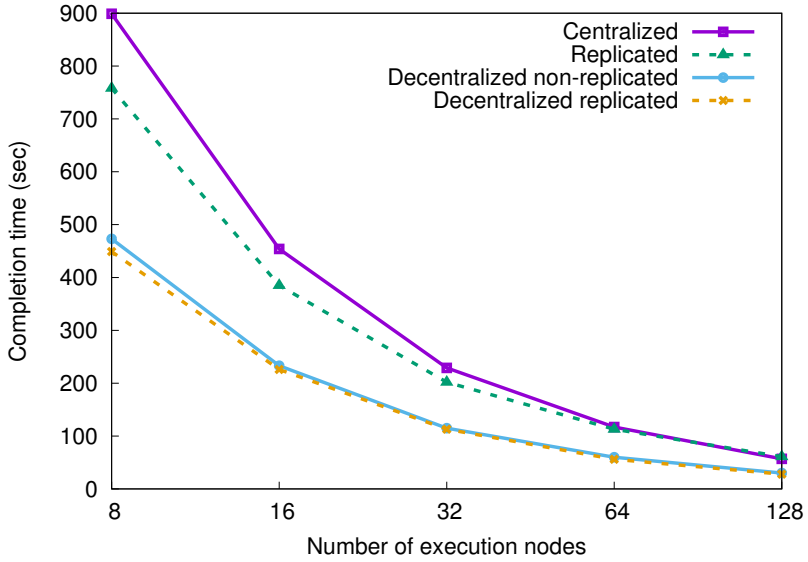


Figure 7.6 – Completion of 32 000 operations as the set size grows.

7.2.3 Support for Real-Life Workflows

The final set of experiments focus on the benefits brought by our strategies to real-life scientific workflows with representative data access patterns: Buzz workflow [43] is a near-pipelined application (Figure 7.2). On the other hand, Montage [16] includes flow splits, parallelized jobs and merge operations (Figure 7.1). We generated synthetic simulations of each of them. Tasks internal computation was emulated by defining an *idle* period for each node. The workflow jobs were evenly distributed across 32 nodes of the four datacenters. We covered three execution scenarios: small scale, computation intensive and metadata intensive. Table 7.1 summarizes their settings.

Scenario	**Small Scale**	**Comp. Int.**	**Metadata Int.**
Operations / node	100	200	1 000
Computation time / node	1 s	5 s	1 s
Total ops - BuzzFlow	7 200	14 400	72 000
Total ops - Montage	16 000	32 000	150 000

Table 7.1 – Settings for real-life workflow scenarios.

In Figure 7.7 we compare the makespan of our strategies in the three above scenarios. We firstly confirm that at small scale a decentralized approach actually adds overhead to the computation and hence centralized solutions are best for *small* settings, regardless of the workflow layout (pipelined or parallel). We note as well that in computation intensive workflows the low metadata interaction benefits *centralized* replication while penalizing *distributed* replication, because the latter is optimized for metadata intensive workloads. Overall, we assert that our decentralized solutions fit complex workflow execution environments, notably metadata intensive applications, where we achieved a 15 % gain in a near-pipeline workflow and 28 % in a parallel, geo-distributed application compared to the centralized baseline.

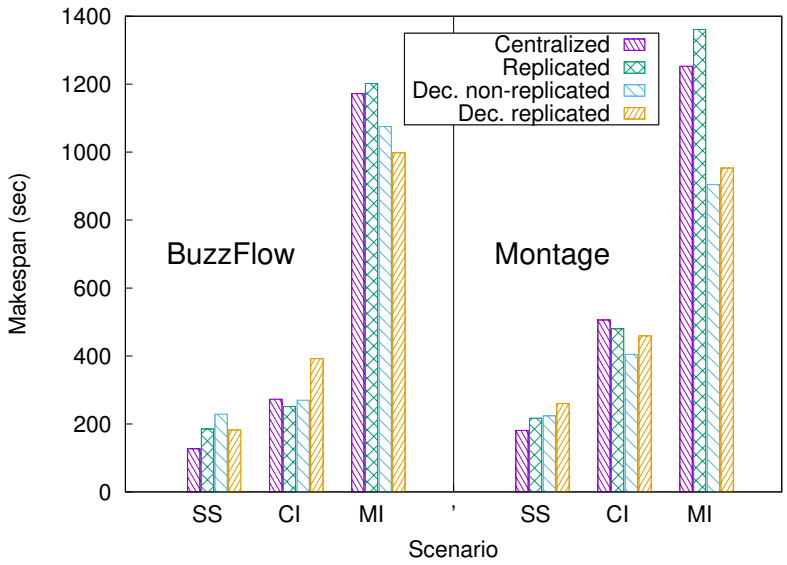


Figure 7.7 – Makespan for two real-life workflows. SS — Small Scale, CI — Computation Intensive, MI — Metadata Intensive.

7.2.4 Discussion

File metadata for early data provisioning. Our strategies enable metadata to be propagated as soon as the data is created. In a scientific workflow such metadata includes file location(s) information. During a multisite workflow execution several tasks are scheduled simultaneously in different locations and require pieces of distributed data to start running. Thanks to our solution, tasks would learn about remote data location early enough and could request the data to be streamed as it is being generated, reducing the costly transfer-related idle time.

Low overhead with eventual consistency. Our approach guarantees that each metadata operation takes effect instantaneously when it happens in the local datacenter, and is propagated to remote datacenters with the lowest possible delay. Other distributed storage tools, such as Facebook's TAO graph storage [22], continue to adopt *eventual* over *strong consistency* in the understanding that, in the practice, the CAP theorem[6] rules the implementation of shared-data distributed systems [21]. Relaxing consistency requirements would lead to a higher availability, which will benefit the execution time, as we observed in our experiments.

In this section we have explored means to better hide latency for metadata access as a way of improving a workflow's global performance in multisite environments. Our solutions efficiently handle file metadata for a large number of small data units. In the next section, we look at the metadata properties to find ways to further boost multisite execution. In particular, we focus on metadata *access frequency*.

[6]CAP — Consistency, Availability, Partitions. In short: You can have at most two of these properties for any shared-data system.

7.3 Separate Handling of Hot and Cold Metadata

The second set of experiments relates to the implementation of *hot* metadata filtering, discussed in Chapter 4. Our hot metadata management strategies are coupled with DMM-Chiron, a multisite workflow execution engine presented in Section 6.2.

Cloud Configuration

DMM-Chiron was deployed on Azure cloud, using a total of 27 nodes of *standard_A4* virtual machines (8 cores, 14 GB memory). The VMs were evenly distributed among three datacenters: West Europe (WEU), North Europe (NEU), and Central US (CUS). Control messages between master nodes are delivered through the Azure Bus.

7.3.1 Hot Metadata for Different Workflow Structures

Our hypothesis is that no single decentralized strategy can fit all workflow structures: a highly parallel task would exhibit different metadata access patterns than a concurrent data gathering task. Thus, the improvements brought to one type of workflow by either of the strategies might turn to be detrimental for another. To evaluate this hypothesis, we ran several combinations of our strategies with the featured workflows. We recall the three strategies presented in Section 4.3.3: local without replication (LOC), hashed without replication (DHT), hashed with local replication (REP), and a centralized baseline (CEN).

Figure 7.8 shows the average execution time for the Montage workflow generating 0.5-, 1-, and 2-degree mosaics of the sky, using in all the cases a 5.5 GB image database distributed across the three datacenters. With a larger number of degrees, a larger volume of intermediate data is handled and a mosaic of higher resolution is produced. Table 7.2 summarizes the volumes of intermediate data generated per execution.

	0.5-degree	1-degree	2-degree
CEN	1.4 (0.5, 0.5, 0.4)	4.9 (1.7, 1.5, 1.7)	17.1 (6.0, 6.4, 4.7)
LOC	1.3 (0.7, 0.2, 0.4)	4.8 (2.1, 1.0, 1.7)	16.2 (8.4, 4.6, 3.2)
DHT	1.5 (0.6, 0.6, 0.4)	4.9 (1.9, 1.3, 1.8)	16.6 (5.4, 4.9, 6.2)
REP	1.4 (0.5, 0.5, 0.4)	4.9 (1.5, 1.9, 1.5)	16.8 (6.6, 3.8, 6.4)

Table 7.2 – Intermediate data in GB for Montage executions using different number of degrees. Per-site breakdown is expressed as: Aggregated (size WEU, size NEU, size CUS).

In the chart we note in the first place a clear time gain of up to 28 % by using a local distribution strategy instead of a centralized one, for all the degrees. This result was expected since the hot metadata is now managed in parallel by three instances instead of one, and it is only the cold metadata that is forwarded to the coordinator site for scheduling purposes (and used at most one time).

We observe that for 1-degree mosaics and smaller ones, the use of distributed hashed storage also outperforms the centralized version. However, we note a performance degradation in the hashed strategies, starting at 1 degree and getting more evident at 2 degrees. We attribute this to the fact that there is a larger number of long-distance hot metadata operations compared to the centralized approach: with hashed strategies, 1 out of 3 operations

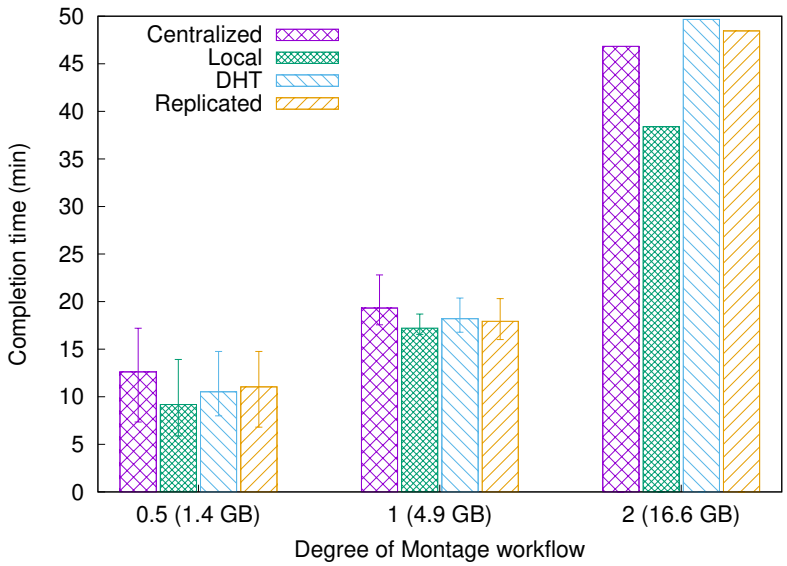


Figure 7.8 – Montage workflow execution time for the different strategies and number of degrees. Average intermediate data shown in parenthesis.

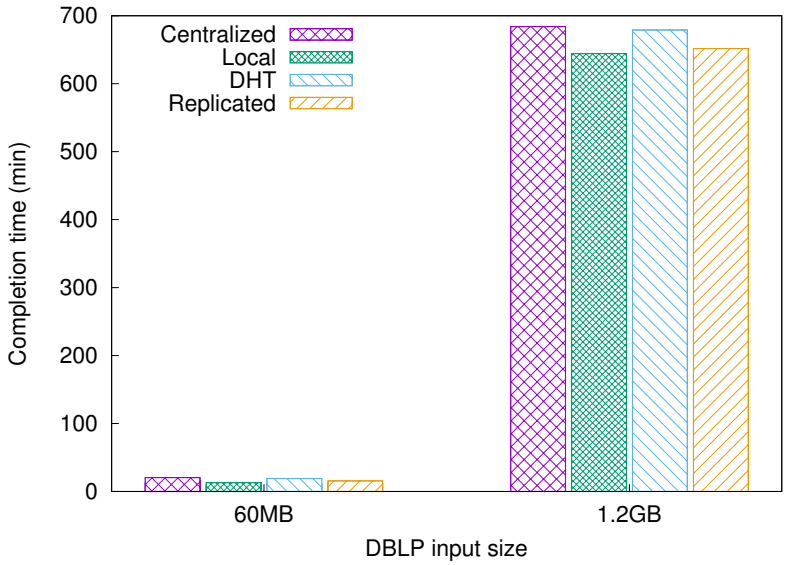


Figure 7.9 – Buzz workflow execution time. Left Y-axis scale corresponds to 60 MB execution, right Y-axis to 1.2 GB.

are carried out on average between CUS and NEU. In the centralized approach, NEU only performs operations in the WEU site, thus such long latency operations are reduced. We also associate this performance drop with the size of intermediate data being handled by the system. We try to minimize inter-site data transfers; however, with larger volumes of data such transfers affect the execution time up to a certain degree and independently of the metadata management scheme.

We conclude that while the DHT method might seem efficient due to linear read and

write operations, it is not well suited for geo-distributed executions, which favor locality and penalize remote operations.

In a similar experiment, we validated DMM-Chiron using the Buzz workflow, which is rather data intensive, with two DBLP database dumps of 60 MB and 1.2 GB. The results are shown in Figure 7.9; note that the left and right Y-axes differ by one order of magnitude. We observe again that DMM-Chiron brings a general improvement in the completion time with respect to the centralized implementation: 10 % for LOC in the 60 MB dataset and 6 % for 1.2 GB, while for DHT and REP the time improvement was less than 5 %.

In order to better understand the performance improvements brought by DMM-Chiron, and also to identify the reason of the low runtime gain for the Buzz workflow, we evaluated Montage and Buzz in a per-job granularity. The results are presented in the next section. Albeit the time gains perceived in the experiments might not seem significant at first glance, two important aspects must be taken into consideration.

Optimization at no cost. Our proposed solutions are implemented using exactly the same number of resources as their counterpart centralized approaches: the decentralized metadata stores are deployed within the master nodes of each site and the control messages are sent through the same existing channels. This means that such gains (if small) come at no additional cost for the user.

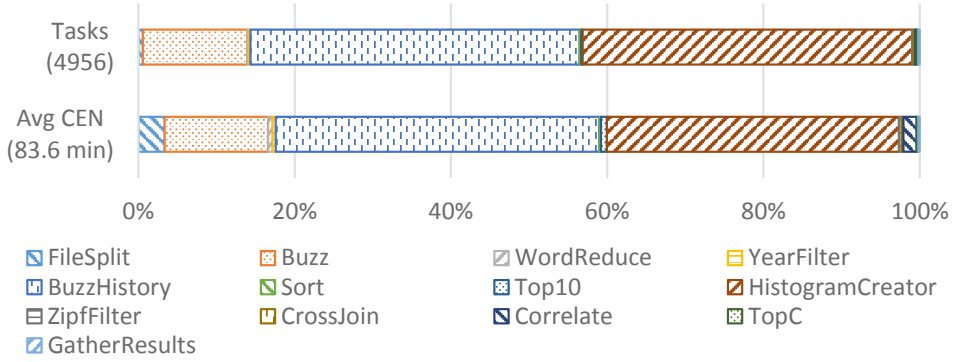
Actual monetary savings. Our longest experiment (Buzz 1.2 GB) runs in the order of hundreds of minutes. With today's scientific experiments running at this scale and beyond, a gain of 10 % actually implies savings of hours of cloud computing resources.

7.3.2 Zoom on Multi-Task Jobs

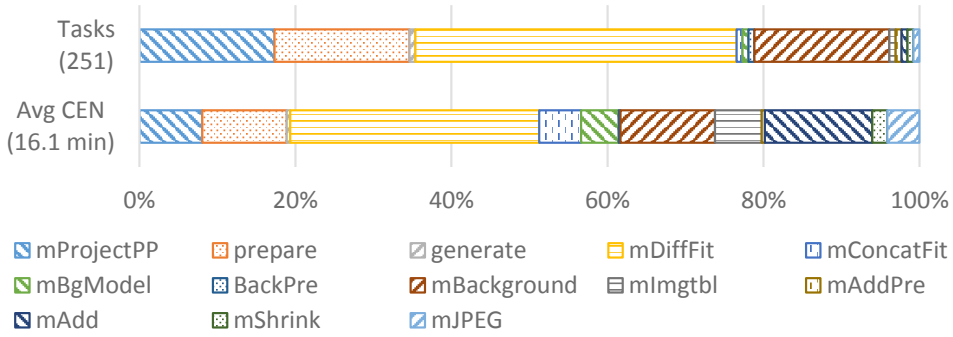
We call a job *multi-task* when its execution consists of more than a single task. In DMM-Chiron, the various tasks of such jobs are evenly distributed to the available sites and thus can be executed in parallel. We argue that it is precisely in these kind of jobs that DMM-Chiron yields its best performance.

Figure 7.10 shows a breakdown of Buzz and Montage workflows with the proportional size of each of their jobs from two different perspectives: tasks count and average execution time. Our goal is to characterize the most *relevant* jobs in each workflow by number of tasks and confirm their relevance by looking at their relative execution time. In Buzz, we notice that both metrics are highly dominated by three jobs: Buzz (676 tasks), BuzzHistory (2134) and HistogramCreator (2134), while the rest are so small that they are barely noticeable. FileSplit comes fourth in terms of execution time and it is indeed the only remaining multi-task job (3 tasks). Likewise, we identify for Montage the only four multi-task jobs: mProject (45 tasks), prepare (45), mDiff (107) and mBackground (45).

In Figures 7.11 and 7.12 we look into the execution time of the multi-task jobs of Buzz and Montage, respectively. Figure 7.11 corresponds to Buzz SWf with 60 MB input data. We observe that except for one case, namely Buzz job with REP, the decentralized strategies outperform considerably the baseline (up to 20.3 % for LOC, 16.2 % for DHT and 14.4 % for REP). In the case of FileSplit, we argue that the execution time is too short and the number of tasks too small to reveal a clear improvement. However, the other three jobs confirm that DMM-Chiron performs better for highly parallel jobs. It is important to note that these gains are much larger than those of the overall completion time (Figure 7.2) since there are still a



(a) Buzz



(b) Montage

Figure 7.10 – Workflow per-job breakdown. Very small jobs are enhanced for visibility.

number of workloads executed sequentially, which have not been optimized by the current release of DMM-Chiron.

Correspondingly, Figure 7.12 shows the execution of each multi-task job for the Montage SWf of 0.5 degree. The figure reveals that, on average, hot metadata distribution substantially improves centralized management in most cases (up to 39.5 % for LOC, 52.8 % for DHT and 64.1 % for REP). However, we notice some unexpected peaks and drops specifically in the hashed approaches. After a number of executions, we believe that such cases are due to common network latency variations of the cloud environment added to the fact that the execution time for the jobs is rather short (in the order of seconds).

7.3.3 Discussion: Limitations to be Addressed

Reduce inter-site data transfers. Our focus in this section was on handling metadata in a smart distributed way so that this improves job/task execution time when processing a large number of data pieces. While our techniques show an improvement with respect to a centralized management, we also notice that when the scale of the SWf and the size of data become larger, there is a degradation in the performance of DMM-Chiron (see Figure 7.8) due to the increase of intermediate data transfers. To mitigate this degradation and allow for larger datasets, DMM-Chiron should be adapted by adding data location awareness to the interface between the *Multisite Transfer* module and the *Metadata Manager*.

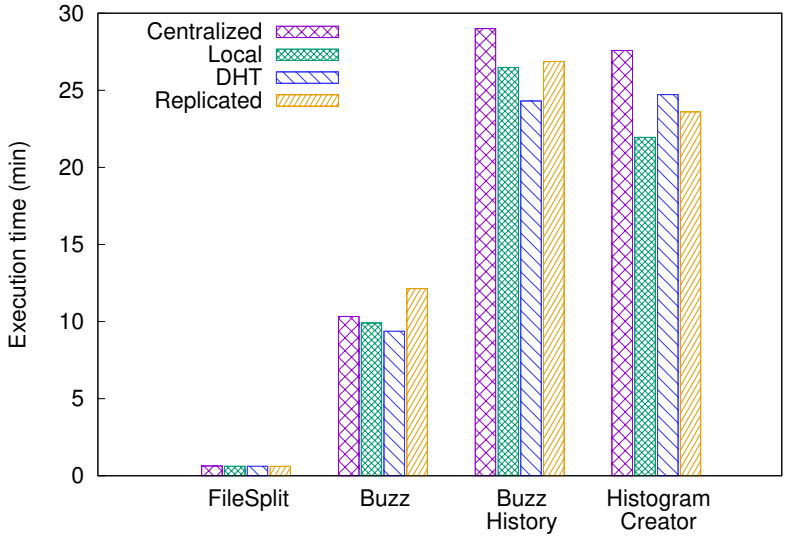


Figure 7.11 – Execution time of multi-task jobs on the Buzz workflow (60 MB input data).

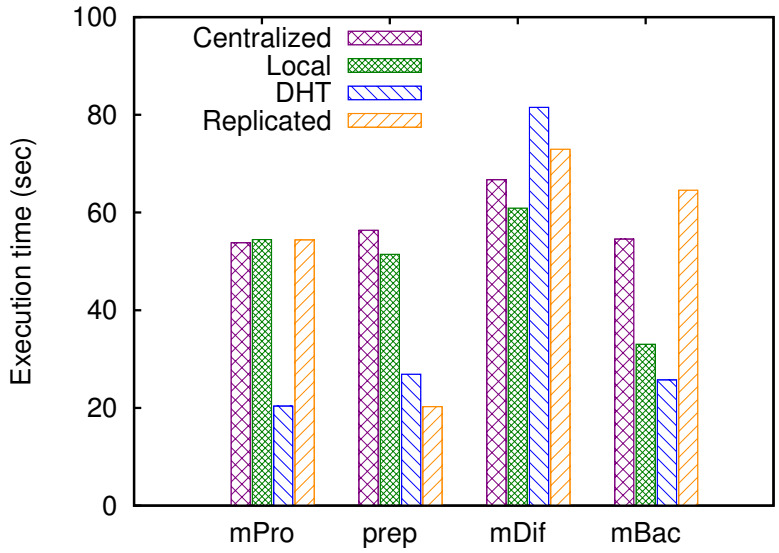


Figure 7.12 – Execution time of multi-task jobs on the Montage workflow of 0.5 degrees.

Load balancing in heterogeneous environments. We have so far only considered the case of a homogeneous multisite environment, where each site has the same amount of VMs, with the same size and capacity. While this kind of configuration is often utilized in workflow execution, in several cases multisite clouds are heterogeneous (different number of resources, different VM sizes). A next step in this path would be to account for these variations in order to balance the hot metadata load according to each site's computing capacity.

This section concludes the experiments related to metadata management on multiple Azure Cloud datacenters. Next, we report on observations for elasticity triggers in applications running on Chameleon Cloud.

7.4 Identifying Thresholds for Elastic Scaling

This third and last set of experiments differs significantly from the previous two. In this section we execute a real workflow application with the goal of pinpointing some application characteristics that could incite elasticity along the execution. As we described in Chapter 5, identifying these elements will allow to model and predict elastic behavior.

Cloud Configuration

We used two bare-metal nodes on Chameleon cloud. Each node has 24 physical cores (48-threads, when hyperthreading is enabled) and 128 GB memory, connected by a 10 Gb/s network. Our use case, the DIBBs workflow (Figure 7.13) is composed by Pig scripts (Filtering stage) and MapReduce jobs (Classification stage), which were executed on Apache Hadoop 2.7 (YARN), using the default HDFS block size of 128 MB as the baseline, unless otherwise stated. Along the section we use the term *container*; a container is YARN's execution unit, with a specific amount of resources allocated (memory, CPU). By default, we assigned 1 GB memory and 1 vCPU per container. Twitter data are provided in raw files where each line represents a tweet, either geo-tagged or not. Each file represents a one-day window, weights ~1.9 GB and contains 3.2 million tweets on average. We used up to 42 days of data.

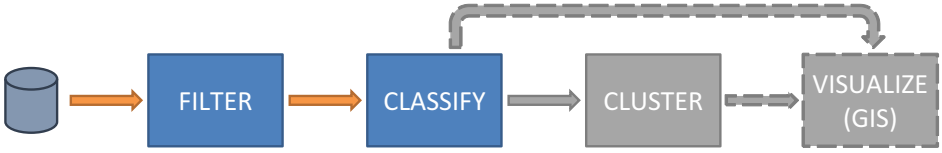


Figure 7.13 – DIBBs workflow for spatial data synthesis. Unavailable jobs are grayed out.

7.4.1 Exploring Tradeoffs Related to Data Block Size

The first task was to assess the tradeoffs of using different block sizes for data storage and manipulation, in order to determine the right configuration of an appliance at the moment of provisioning, according to the data size to be processed. This evaluation required two experiments. First, we looked into the impact of block size for uploading the data from local disk to HDFS (corresponding to a command `hadoop fs -put`), we used from 5 to 20 days of data (up to 37 GB) while varying the block size from 128 MB (default, baseline) to 2 GB. The results are displayed in Figure 7.14. We observe up to 70 % improvement in upload time when using 2 GB block size with respect to the baseline. Intuitively, this would suggest that the right block size to use is the largest size that a node can store. However, a second evaluation challenges this intuition.

We complement the experiment by looking at the average filtering time for different dataset sizes while varying the block size from 64 MB to 1024 MB. We set YARN to work with 96 containers, 48 per node to utilize all the available threads. As we note in Figure 7.15, the filtering takes longer with a smaller block size, we attribute this behavior to the fact that the number of containers is exceeded by the number of blocks, therefore some blocks have to be queued for later processing, thus extending the execution time. On the other hand, we

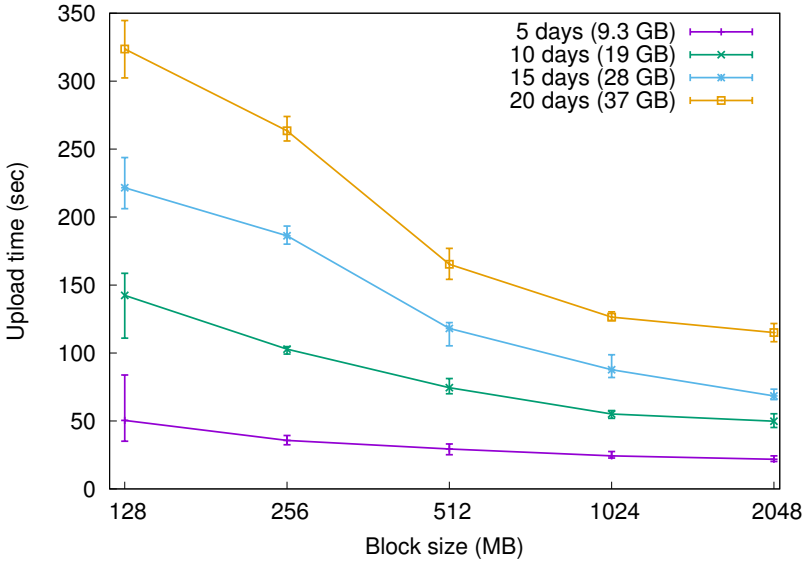


Figure 7.14 – Data uploading time per HDFS block size.

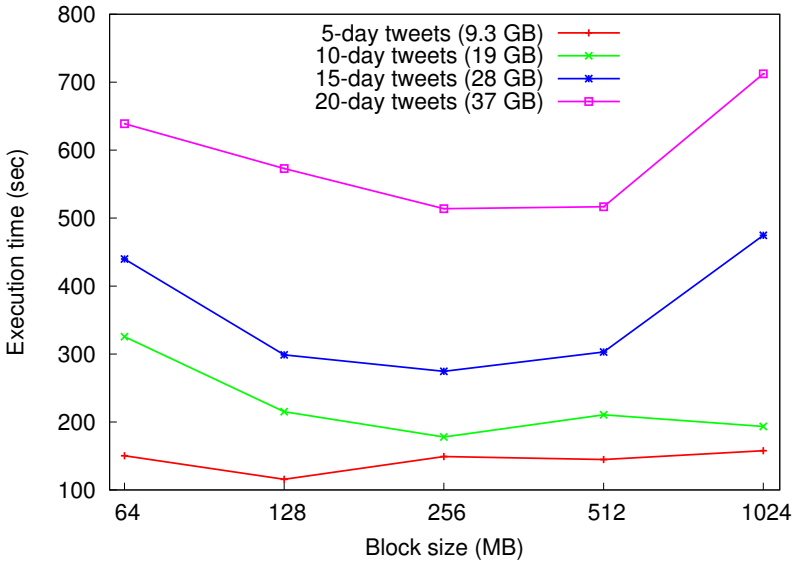


Figure 7.15 – Average execution time for filtering the tweets of the State of Illinois.

also notice that larger block sizes slow down the application; we assert that this is due to an underutilization of the containers (less blocks than containers) and, at the same time, each container deals with a larger chunk of data. It is an intermediate block size which hits the *sweet spot* of performance in most cases.

Therefore, we conclude that the block size should be tuned in a way that the block count maximizes the utilization but does not exceed the number of processing units (*containers*, in this scenario).

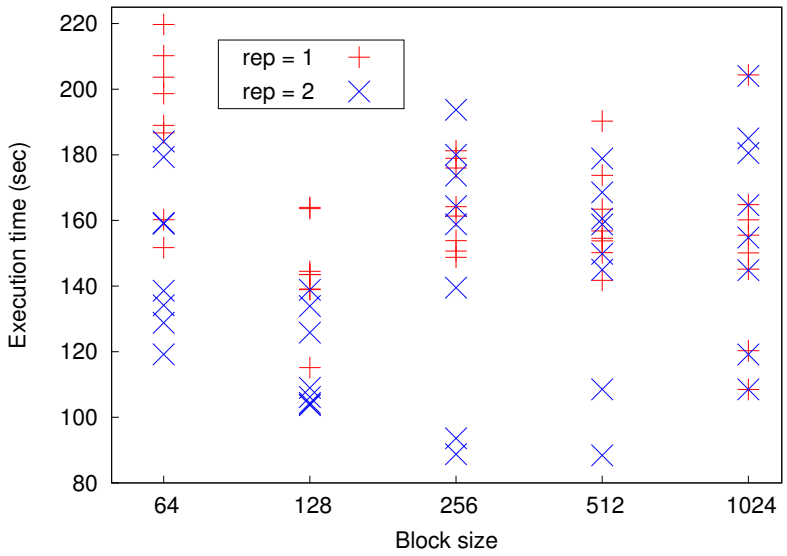


Figure 7.16 – Replication factor for 5-day (9.7 GB) tweet filtering.

7.4.2 Impact of the Data Replication Factor

Our second parameter to evaluate is *data replication*. Many file systems offer the possibility to enable data replication, both for data locality and for fault tolerance. Our goal is to understand what are the benefits brought by replicating data and whether they justify the use of additional storage resources. For this experiment we use HDFS replication factor of 1 (no replication) and 2 (single replication) in our two bare-metal nodes. In the first case, this means that the whole dataset is divided between the two nodes. With single replication, each node holds a full copy of the dataset, divided in blocks.

We first executed the tweet filtering component, using a small dataset of ~10 GB, corresponding to five days of tweets. We ran the script eight times for different block sizes, from 64 MB to 1 GB. As shown in Figure 7.16, there is no clear performance improvement brought by data replication. While some executions were faster with replication factor of 2, the results were not consistent and in many cases the no-replication runs outperformed their counterparts.

Next, we tested the application with larger datasets to observe the disk usage along the execution when using data replication. The data size ranged from 67 GB to 79 GB (36 to 42 days of tweets), using the default block size of 128 MB and the available disk of 256 GB. Roughly 30 GB of the disk are in use for general purpose (OS, software, etc.), the rest is used for the application's data. Figure 7.17 shows the progress of disk saturation with respect to the elapsed execution time. In all four cases the application generates a significant amount of intermediate data (always replicated), going beyond 180 GB. The first three runs, successfully complete their execution, releasing all the disk space used for intermediate data. The difference in disk usage between the beginning and end of the run, shows that about 30 GB (~40 % the size of the input data) were generated as output data.

The largest run (42 days), however, reaches almost the limit of disk space and fails to finish, also failing to release intermediate data generated. We attempted new executions with the same and larger data sizes and we obtained the same result. The failure is due to YARN's

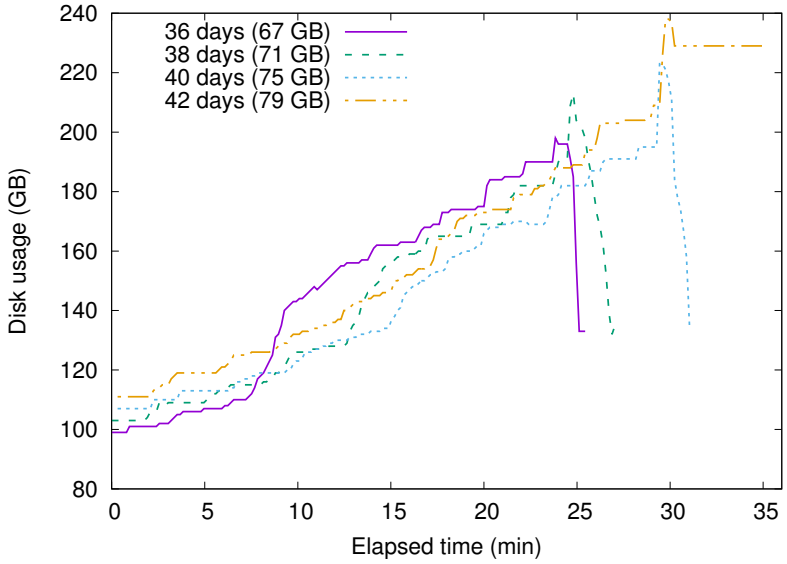


Figure 7.17 – Disk space occupied along the execution of tweet filtering (disk size: 256 GB).

health checker, which establishes a maximum disk utilization (90 % by default) in order to prevent the whole environment from crashing. Nevertheless, YARN does not provide a graceful termination of the application that clears the intermediate data, it rather waits idle for some disk space to be freed, as we can see from the horizontal line at the end of the execution. This situation blocks the disk and forces the user to manually perform the release. The scenario is not specific to YARN, other systems would not only stay idle, but even crash when the disk capacity is reached. A key characteristic of cloud computing is a managed environment that allows the user to focus on their business goals. In our experiment this aspect is not satisfied, as users have to "clean their mess" when the application fails.

To replicate or not to replicate? Data replication implies that input, intermediate and output data will be duplicated. In our example, a dataset of ~80 GB easily saturated a disk with capacity of three times its size. We did not notice a significant performance improvement with data replication, but rather a disk prone to saturation. We therefore conclude that *it is preferable to avoid data replication by default*. This could only be overridden by policies acknowledged and approved by the user, in justified scenarios where data backups or data locality are crucial.

Furthermore, this experiment perfectly illustrated the motivation for *smart* scaling: in the run of 42 days, only few additional gigabytes were needed for the execution to be completed few minutes later. In a classic elastic scenario, a new *big* node would have been provisioned to finish the task, we recall that Chameleon nodes featured 24 cores, 128 GB memory and 256 GB disk (and leased per hour!). With *smart elastic scaling*, modelling the application could help us to select a much smaller machine and save cloud resources.

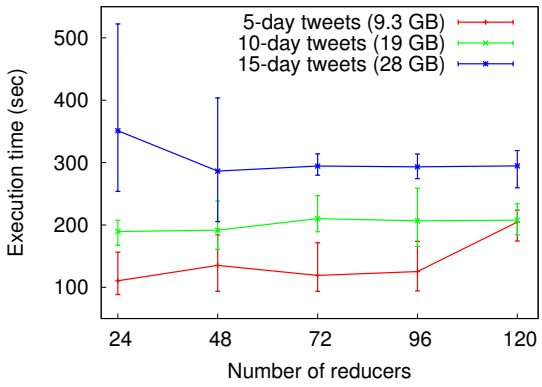


Figure 7.18 – Increasing number of reducers (block size 256 MB).

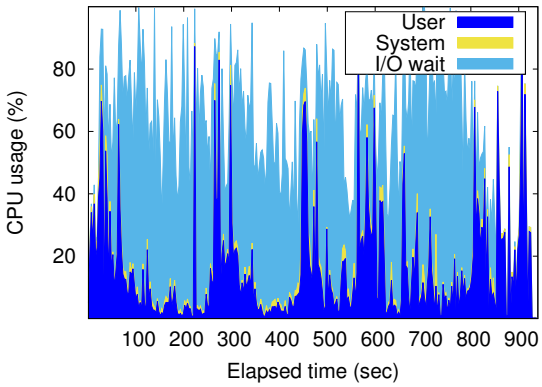


Figure 7.19 – CPU usage of 1-day filtering for Chicago area.

7.4.3 Parameters Not Modelled

Other parameters were part of our experimental study. However, they yielded undesired behaviors which make them hard to model. We briefly report our observations regarding two of them: number of reducers and CPU usage.

Number of reducers. In YARN, the number of *mappers* is determined by the number of blocks obtained after splitting the input data. On the other hand, the default number of *reducers* can be overridden as a configuration parameter. With this in mind, we intended to model the performance of the application with respect to the number of reducers running concurrently; our expectation was to see a performance improvement when all the containers were used. We ran tests for three different data sizes, varying the reducers in multiples of the available cores per node (24), the total number of containers in the cluster is 96; the results are shown in Figure 7.18. We noticed a mostly steady behavior regardless of the number of reducers, especially for a 10-day sample. For the 5-day sample we noticed an overall slight decline in the performance with more reducers (lower is better in the plot); whereas for 15-days we observed a slight increase.

Unfortunately, the lack of a clear correlation prevented us from using this parameter for modelling. We inferred that these low variations are because the application is not computing- but rather data-intensive and so adding computing units does not impact the overall execution time. To verify this, we monitored the CPU activity as shown below.

CPU usage. To understand our application's behavior, in search for additional parameters to model, we plotted the CPU usage for a simple execution of 1-day filtering using a single node (Figure 7.19). We validated that, indeed, most of the CPU time is spent on *I/O wait*, with an average of ~50 % and peaks of up to 95 %. The Linux User's Manual of the `top` command, defines IO-wait as "*time [the CPU spends] waiting for I/O completion*" [69]. Since the workflow is the sole user application running on the node, the actual application CPU usage corresponds only to the *User* percentage of the plot.

We observed that for this class of data-intensive applications, where the execution time is dominated by I/O operations, it is unpractical to use CPU time measurements to model

the performance or the behavior of the application. We believe that the focus should then be on disk-related parameters, as we did before with data uploading and disk saturation.

7.4.4 Discussion: *Smart* Policies for *Smart* Provisioning

We have identified parameters from workflow jobs that can be modelled to predict the behavior of data-intensive applications. We have also recognized other factors that do not correlate well to the workflow's execution, and thus cannot be easily modelled. A logical next step will be to implement smart policies driven by the observations presented above, and by other similar parameters yet to be pinpointed. They would be incorporated to the collection of policies that steer the decision engine in our proposed smart elastic provisioner (Section 5.2.2).

What is a smart policy? A typical policy for elastic provisioning would decide to create a new machine once the current capacity of a resource goes beyond a threshold [12]. A smart policy, based on a statistical model of the application, would evaluate other factors to determine what kind of machine should be provisioned, or whether it should be provisioned at all. These factors include the following.

Forecasted time to completion. If the remaining execution time is short, an application might have finished by the time a new machine is available. Nevertheless, users will be charged by the newly-created, unused VM. A smart policy could prevent this from happening.

Expected storage required. The application might need only few gigabytes of additional storage to finish (like in Figure 7.17). A smart policy would spot this situation and decide to provision a smaller machine.

Input or aggregated data size. Knowing the data size would help to tune application-specific parameters in the smart appliance for a better performance of the machine to provision. This is the scenario observed in the present section.

Our experimental evaluation revealed a motivating scenario where smart elastic scaling could have optimized resource usage. In the case of Chameleon, as in other private clouds, resource optimization means that more cores and storage are available for other users. Moreover, in public clouds, resource optimization could be translated into important monetary savings at a large scale.

7.5 Conclusion

In this chapter we demonstrated the efficacy of our strategies for multisite metadata management for scientific workflows. Through a series of experiments, we have validated several scenarios where they outperform the traditional centralized approaches. More importantly, this evaluation helped us to determine under which circumstances our proposal is most effective (and under which it is not).

First of all, we proved that a hybrid distributed/replicated solution can reduce, almost by half, the time to process inter-site metadata operations. We also showed that these solutions scale to hundreds of nodes. Then, we found a best-match between our decentralized strategies and metadata-intensive, large-scale workflows. At the same time, we acknowledged the prevailing effectiveness of centralized metadata servers for single-site or smaller-scale workflows.

Moreover, our proposal and implementation of a separate management for *hot* and *cold* metadata further improved the multisite execution time of Montage and BuzzFlow, our use cases. Additionally, we discovered that this metadata filtering technique reaches its best efficiency in *multitask* jobs.

In an orthogonal direction, we detected parameters to steer elastic provisioning decisions in a workflow application for geospatial analysis running on a private cloud. This real-world application, provided us with a true feeling of the issues faced when handling scientific workflows, beyond our previous experimental, synthetic and controlled environments.

Part IV

Conclusions and Perspectives

Chapter 8

Conclusions

Contents

8.1 Achievements **94**

8.2 Future Work and Perspectives **95**

In the context of fast growing volumes of data to be processed at larger and larger scales, scientific workflows are emerging as a natural data processing paradigm. Workflows help defining data dependencies between the different components of an application. Scientific workflow management systems (SWfMS) provide tools to configure, run and monitor them, allowing for concurrent execution of non-dependent jobs.

Workflows were conceived for HPC systems, but also became popular on clouds to take advantage of cloud-specific capabilities like elasticity and portability. SWfMS run on HPC-like cloud infrastructures, consisting of virtual clusters with a shared file system in a single datacenter. However, increasing storage and processing requirements of large-scale workflows, together with quota limitations imposed by cloud providers, force workflows to be distributed to several datacenters (multisite workflows).

The execution of a multisite workflow faces several limitations. There is no longer a common file system spanning across all sites, so a new way of data transfer should be adapted. Also, inter-site latencies impact the workflow performance, as even small control messages or metadata updates take much longer to be transmitted.

In this thesis, we have provided different strategies for improving the performance of workflows on multisite clouds, targeting two challenges. The first and main challenge is *metadata management*; in this line we proposed a decentralized metadata manager, and a protocols for hot metadata handing. The second challenge is *elastic resource usage*; in order to tackle it, we collaborated in a study to detect and model parameters that elicit elastic scaling.

8.1 Achievements

In this section we summarize the achievements obtained in each of the three contributions of this thesis, in an attempt to provide efficient support for multisite workflows.

8.1.1 Enabling Multisite Metadata Management for Workflows

As of today, state-of-the-art public clouds do not provide adequate mechanisms for *efficient metadata management* across datacenters for scenarios involving masses of geo-distributed data that are stored and processed in multiple sites across the planet. In this direction, we have investigated approaches to metadata management enabling an efficient execution of such *geographically distributed workflows running on multisite clouds*. We focused on a common scenario where workflows generate and process a huge number of small files, which is particularly challenging with respect to metadata management. As such workloads generate a deluge of small and independent I/O operations, efficient metadata handling is critical.

To address this problem, we explored means to better hide latency for metadata access as a way of improving the global performance. We proposes specific techniques that implement this approach, combining distribution and replication for in-memory metadata partitioning. Although such techniques are already known by the community of distributed data management, to the best of our knowledge, our work is *the first attempt to bridge the gap between single- and multisite cloud metadata management*. Our solution leverages both the workflow semantics (e.g. control-flow patterns) and the practical tools available on today's public clouds (e.g. caching services on PaaS clouds) to propose several strategies for decentralized metadata management.

We have studied the performance of these techniques in a consistent environment by implementing them all in the Azure Cloud. Our set of design principles enable an unprecedented metadata access performance for workflows scattered across widely distributed cloud sites.

8.1.2 Managing Hot Metadata Across Datacenters

We have introduced the concept of *hot metadata* (frequently accessed metadata) for scientific workflows running in large, geographically distributed and highly dynamic environments. Based on this concept, we designed a hybrid decentralized and distributed model for handling hot metadata in multisite clouds. It relies on a metadata filtering component implemented on top of a multisite workflow engine.

Our proposal was able to optimize the access to and ensure the availability of hot metadata, while effectively hiding the inter-site network latencies and remaining non-intrusive and easy to deploy. Compared to state-of-the-art centralized solutions, our strategies propitiated a significant improvement for the whole workflow's completion time (beyond 20 %). Specifically, our implementation managed to reduce by half the execution time for highly-parallel workflow jobs. These gains come at no additional monetary cost, since the metadata storage is implemented within the same nodes of the workflow management system.

To the best of our knowledge, our work represents *an unprecedented mechanism for handling hot metadata* both in a workflow management system and in a multisite cloud environment.

8.1.3 Studying Elasticity Parameters to Steer Smart Scaling

Thanks to a real-world workflow for spatial data synthesis, we were able to identify application parameters that are involved in the decision for elastic provisioning in a cloud. The goal of the project is to be able to model these parameters and generate policies to drive smart scaling. We observed that the data block size is inversely proportional to the time taken to load the data into the system. At the same time, large blocks produce slower performance in some nodes while leaving others in an idle state. Additionally, we assessed the tradeoffs of data replication with respect to the disk capacity of each node. We determined that for large scale applications, replication might saturate the resources quickly and should only be encouraged in specific cases of sensitive data. We also detected that in large-scale, data-intensive applications, where read and write operations dominate the overall execution time, measurements like CPU cannot be easily correlated to the applications performance, because they represent a small fraction of the execution time.

The study will allow to *produce statistical models to predict the behavior of workflows applications on clouds*. Later, these models could be turned into policies that help determine the right resource to provision in an smart elastic system. This collaboration enabled us to have a first-hand experience with a real application, and recognize additional parameters that come into play during a workflow execution, several of which we had not looked at before.

8.2 Future Work and Perspectives

In this section we first summarize the potential courses of action discussed along the thesis, in terms of multisite metadata management and implementation of a smart elastic provisioner. Then, we briefly discuss a direction of workflow management: streaming processing.

8.2.1 Improving Multisite Metadata Management

Dynamic hot metadata. Our approach for selecting hot file and execution metadata was based on statistical observations from historical data. A logical question raised by the scientific community was *what about* dynamic *hot metadata?* As we discussed in Chapter 4, there are several ways to achieve this dynamic quality. The simplest approach being to create tags to define hot metadata elements using the workflow's definition language. Another option would be to identify hot metadata by running training executions. However, a truly dynamic solution would be to maintain a rank on the frequency of access to the metadata alongside the execution. This idea of online rankings has been studied for hot data identification. Such concept could adopted and implemented in our hot metadata manager.

Data-location awareness. Our experimental evaluation revealed that proportionally large amounts of intermediate data were generated. The multisite scheduler distributes jobs in a best effort basis at the beginning of the execution. Due to workflows job scattering patterns, data are often required in a different location from where it is created, and needs to be transferred. In Chapter 7 we mentioned the possibility of adding data-location awareness to the scheduler of DMM-Chiron. This location awareness, particularly at the intra-site level would allow tasks to be scheduled at, or closer to the site where the data resides, reducing execution delays due to inter-site data transfers.

8.2.2 Enabling Smart Elastic Provisioning

Implementing a smart provisioning system. Given the short time that we had to develop this idea, there remain open issues in this direction. An additional parameter that could be evaluated and modelled as well is the number of concurrent users of an application, which is the second factor that triggers elastic provisioning. The next step, after the identification and characterization of the mentioned parameters, is to devise functions to model the behavior of the application based on those parameters. Afterwards, these models could be adapted to Phantom provisioner as smart policies for elastic scaling. Finally, a more extensive project would be required in order to design the interfaces for the different components of the proposed system (Section 5.2.2). The portal (CyberGIS Gateway), the provisioner (Phantom), the storage (Chameleon) and the appliance repository are separate pieces that already exist as standalone middleware/infrastructures.

Metadata-driven elastic provisioning. Along the thesis we made evident the importance of file metadata to facilitate multisite workflow execution. File metadata include information about the location(s) of the file or data fragment. Metadata are much less expensive to transfer than the data itself. An efficient propagation of file metadata will allow the system coordinator (whether a master node, or a provisioner) to have a general view of the file system early enough. As presented in Chapter 5, adding data location awareness will allow the elastic provisioner to make decisions on where to provision a new resource, so that it is close to or at the data location. This could be achieved by merging our metadata management nodes to each of the cloud sites available to the provisioner. These instances of the metadata registry would keep updated the decision engine with the location of every piece of data during the whole execution.

8.2.3 Workflow Management: From Batch to Stream Processing

Computing paradigms are always evolving in response to technological and human challenges. Today, data are being continuously generated both by humans and devices. Social networks, mobile phones, radars, sensors on connected objects, etcetera, are perpetually producing information. Data processing is no longer a matter of reading blocks of static (batch) files. Instead, data are often presented as sequences of small-sized records, arriving at fluctuating rates (streams). Big data platforms and applications are increasingly providing support for stream processing, and workflows are following the same path. Apache-based applications are leaning towards streaming projects like Spark, Storm or Flink instead or on top of Hadoop, and the trend has just recently started.

Workflow processing greatly benefit from stream processing, as the intermediate "staging" storage can be bypassed and data can be processed at one job as it is being generated by the previous one or by external sources. *Geo-distributed workflows and streaming data will likely come hand by hand*, as most of these data originates from globally dispersed sources. As we discussed in the manuscript, data are being generated faster than they can be processed. Multisite workflows will, at least, eliminate the additional overhead of data transfers, by allowing to process a stream closer to the source. Several solutions are arising to cope both with streaming on workflows (e.g. Kepler [45]) and multisite data streaming (e.g. JetStream [136]). The next generation of scientific workflows is expected to bring together the best of both approaches.

Bibliography

[1] Lior Abraham, John Allen, Oleksandr Barykin, et al. "Scuba: diving into data at Facebook". In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1057–1067.

[2] William Allcock, John Bresnahan, Rajkumar Kettimuthu, et al. "The Globus striped GridFTP framework and server". In: *SC - ACM/IEEE Conference on Supercomputing*. IEEE. 2005, p. 54.

[3] *Alluxio - Open Source Memory Speed Virtual Distributed Storage*. Alluxio Open Foundation. URL: http://www.alluxio.org/ (visited on 2017-03-11).

[4] Pinar Alper, Khalid Belhajjame, Carole A Goble, et al. "Enhancing and abstracting scientific workflow provenance for data publishing". In: *EDBT - EDBT/ICDT Joint Conference*. ACM. 2013, pp. 313–318.

[5] *AWS Direct Connect*. Amazon Web Services. URL: https://aws.amazon.com/directconnect/ (visited on 2017-03-11).

[6] *Amazon Web Services (AWS) | Cloud Computing Services*. Amazon Web Services. URL: https://aws.amazon.com/ (visited on 2017-03-11).

[7] *Elastic Computing Cloud (EC2) - Cloud Server and Hosting*. Amazon Web Services. URL: https://aws.amazon.com/ec2/ (visited on 2017-03-11).

[8] *Amazon Simple Storage Server (S3) - Cloud Storage*. Amazon Web Services. URL: http://aws.amazon.com/s3/ (visited on 2017-03-11).

[9] *Amazon EMR*. Amazon Web Services. URL: http://aws.amazon.com/emr/ (visited on 2017-03-11).

[10] *Global Infrastructure*. Amazon Web Services. URL: http://aws.amazon.com/about-aws/global-infrastructure/ (visited on 2017-03-14).

[11] *What is Streaming Data?* Amazon Web Services. URL: https://aws.amazon.com/streaming-data/ (visited on 2017-03-14).

[12] *AWS | Auto Scaling*. Amazon Web Services. URL: https://aws.amazon.com/autoscaling/ (visited on 2017-03-11).

[13] *Apache Taverna - Genome and Gene Expression*. Apache Taverna. URL: https://taverna.incubator.apache.org/introduction/taverna-in-use/genome-and-gene-expression/ (visited on 2017-03-14).

[14] Roger S Barga, Jonathan Goldstein, Mohamed Ali, et al. "Consistent streaming through time: A vision for event stream processing". In: *CIDR 2007 - Third Biennial Conference on Innovative Data Systems Research*. Asilomar, CA, USA, Jan. 2007. URL: http://cidrdb.org/cidr2007/.

[15] Khalid Belhajjame, Katy Wolstencroft, Oscar Corcho, et al. "Metadata management in the taverna workflow system". In: *CCGRID - IEEE International Symposium on Cluster Computing and the Grid*. IEEE. 2008, pp. 651–656.

[16] G Bruce Berriman, Ewa Deelman, John C Good, et al. "Montage: a grid-enabled engine for delivering custom science-grade mosaics on demand". In: *SPIE 5493 - Optimizing Scientific Return for Astronomy through Information Technologies*. International Society for Optics and Photonics. 2004, pp. 221–232.

[17] S. Bharathi, A. Chervenak, E. Deelman, et al. "Characterization of scientific workflows". In: *WORKS - IEEE Workshop on Workflows in Support of Large-Scale Science*. 2008, pp. 1–10.

[18] *Blue Waters Projects*. Blue Waters. URL: https://bluewaters.ncsa.illinois.edu/science-teams/ (visited on 2017-03-14).

[19] James K. Bonfield and Rodger Staden. "ZTR: a new format for DNA sequence trace data". In: *Bioinformatics* 18.1 (2002), pp. 3–10.

[20] Dhruba Borthakur. *HDFS Architecture Guide*. URL: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html (visited on 2017-03-11).

[21] Eric A Brewer. "Towards robust distributed systems". In: *PODC - ACM Symposium on Principles of Distributed Computing*. Vol. 7. 2000.

[22] Nathan Bronson, Zach Amsden, George Cabrera, et al. "TAO: Facebook's Distributed Data Store for the Social Graph". In: *ATC - USENIX Annual Technical Conference*. 2013, pp. 49–60.

[23] George Carey-Simos. *How Much Data Is Generated Every Minute On Social Media?* 2015. URL: http://wersm.com/how-much-data-is-generated-every-minute-on-social-media/ (visited on 2017-03-14).

[24] Philip H. Carns, Walter B. Ligon, Robert B. Ross, et al. "PVFS: a parallel file system for Linux clusters". In: *Annual Linux Showcase and Conference*. 2000.

[25] Philip Carns, Sam Lang, Robert Ross, et al. "Small-file access in parallel file systems". In: *IPDPS - IEEE International Symposium on Parallel and Distributed Processing*. IEEE. 2009, pp. 1–11.

[26] *ALICE Data Acquisition*. CERN. URL: http://aliceinfo.cern.ch/Public/en/Chapter2/Chap2_DAQ.html (visited on 2017-03-11).

[27] *The Large Hadron Collider*. CERN. URL: https://home.cern/topics/large-hadron-collider/ (visited on 2017-03-14).

[28] *Worldwide LHC Computing Grid*. CERN. URL: http://wlcg.web.cern.ch/ (visited on 2017-03-14).

[29] *Trigger and Data Acquisition System | ATLAS Experiment at CERN*. CERN. URL: http://atlas.cern/discover/detector/trigger-daq/ (visited on 2017-03-14).

[30] *Tracking | CMS Experiment*. CERN. URL: https://cms.cern/detector/identifying-tracks/ (visited on 2017-03-14).

[31] *About Chameleon*. Chameleon Cloud. URL: https://www.chameleoncloud.org/about/chameleon/ (visited on 2017-03-11).

[32] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al. "Bigtable: A distributed storage system for structured data". In: *ACM Transactions on Computer Systems (TOCS)* 26.2 (2008).

[33] Weiwei Chen and Ewa Deelman. "Partitioning and scheduling workflows across multiple sites with storage constraints". In: *Parallel Processing and Applied Mathematics* (2012), pp. 11–20.

[34] *The Zettabyte Era-Trends and Analysis*. Cisco. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html (visited on 2017-03-11).

[35] *Powerful Supercomputing at Economical Prices*. Cray Inc. URL: http://www.cray.com/sites/default/files/resources/XT-AMD-HLRS-09.pdf (visited on 2017-03-14).

[36] *CyberGIS Software Integration for Sustained Geospatial Innovation*. CyberGIS. URL: http://cybergis.cigi.uiuc.edu/ (visited on 2017-03-11).

[37] Jim Daily and Jeff Peterson. "Predictive Maintenance: How Big Data Analysis Can Improve Maintenance". In: *Supply Chain Integration Challenges in Commercial Aerospace: A Comprehensive Perspective on the Aviation Value Chain*. Springer, 2017, pp. 267–278.

[38] Bert De Reyck, Xiaojia Guo, Yael Grushka-Cockayne, et al. *APOC Business Process Reengineering Big Data Study*. Tech. rep. 2016. URL: https://www.eurocontrol.int/sites/default/files/publication/files/apoc-bpr-big-data-final-report.pdf.

[39] Jeffrey Dean and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1 (2008), pp. 107–113.

[40] Ewa Deelman, Yolanda Gil, and Maria Zemankova. *NSF workshop on the challenges of scientific workflows*. Tech. rep. National Science Foundation, 2006. URL: http://www.isi.edu/nsf-workflows06/.

[41] Ewa Deelman, Gurmeet Singh, Miron Livny, et al. "The Cost of Doing Science on the Cloud: The Montage Example". In: *SC - ACM/IEEE Conference on Supercomputing*. IEEE Press, 2008.

[42] Ewa Deelman, Karan Vahi, Gideon Juve, et al. "Pegasus: a Workflow Management System for Science Automation". In: *Future Generation Computer Systems* 46 (2015), pp. 17–35.

[43] Jonas Dias, Eduardo Ogasawara, Daniel De Oliveira, et al. "Algebraic dataflows for big data analysis". In: *BigData - IEEE International Conference on Big Data*. IEEE. 2013, pp. 150–155.

[44] Matthieu Dorier, Gabriel Antoniu, Franck Cappello, et al. "Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O". In: *CLUSTER - IEEE International Conference on Cluster Computing*. IEEE. 2012, pp. 155–163.

[45] Lei Dou, Daniel Zinn, Timothy McPhillips, et al. "Scientific workflow design 2.0: Demonstrating streaming data collections in Kepler". In: *ICDE - IEEE 27th International Conference on Data Engineering*. IEEE. 2011, pp. 1296–1299.

[46] *FAQs | Earthscope*. EarthScope. URL: http://www.earthscope.org/information/faqs/ (visited on 2017-03-11).

[47] *Company Info | Facebook Newsroom*. Facebook. URL: http://newsroom.fb.com/company-info/ (visited on 2017-03-14).

[48] *Video Views: Facebook Video*. Facebook. URL: https://www.facebook.com/business/ads-guide/video-views/ (visited on 2017-03-14).

[49] Brad Fitzpatrick. "Distributed Caching with Memcached". In: *Linux Journal* 124 (Aug. 2004).

[50] Ian Foster, Timothy Freeman, Kate Keahy, et al. "Virtual clusters for grid communities". In: *CCGRID - IEEE International Symposium on Cluster Computing and the Grid*. IEEE. 2006, pp. 513–520.

[51] Luiz MR Gadelha, Michael Wilde, Marta Mattoso, et al. "MTCProv: a practical provenance query framework for many-task scientific computing". In: *Distributed and Parallel Databases* 30.5-6 (2012), pp. 351–370.

[52] Wojciech Galuba and Sarunas Girdzijauskas. "Distributed Hash Table". In: *Encyclopedia of Database Systems*. Springer, 2009, pp. 903–904.

[53] *The 100,000 Genomes Project*. Genomics England Ltd. URL: https://www.genomicsengland.co.uk/the-100000-genomes-project/ (visited on 2017-03-14).

[54] Dan Gibson. *Is Your Big Data Hot, Warm, or Cold?* 2012. URL: http://www.ibmbigdatahub.com/blog/your-big-data-hot-warm-or-cold/ (visited on 2017-03-11).

[55] *Google Cloud Platform*. Google. URL: https://cloud.google.com/ (visited on 2017-03-11).

[56] *Preemptible VMs - Compute Instances*. Google. URL: https://cloud.google.com/preemptible-vms/ (visited on 2017-03-11).

[57] *BigQuery - Analytics Data Warehouse*. Google. URL: https://cloud.google.com/bigquery/ (visited on 2017-03-11).

[58] *Resource Quotas - Google Cloud Platform*. Google. URL: https://cloud.google.com/compute/docs/resource-quotas/.

[59] The Apache Software Foundation. *Apache Hadoop*. URL: http://hadoop.apache.org/ (visited on 2017-03-14).

[60] David Hollingsworth. *Workflow management coalition: The workflow reference model*. Tech. rep. TC00-1003. Document Status - Issue 1.1. Jan. 1995. URL: http://www.wfmc.org/resources/.

[61] *IBM Archives: System/360 Announcement*. IBM. URL: http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PR360.html (visited on 2017-03-14).

[62] *The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, Executive Summary*. International Data Corporation. URL: https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm (visited on 2017-03-11).

[63] Gideon Juve, Ann Chervenak, Ewa Deelman, et al. "Characterizing and profiling scientific workflows". In: *Future Generation Computer Systems* 29.3 (2013). Special Section: Recent Developments in High Performance Computing and Security, pp. 682–692.

[64] Gideon Juve and Ewa Deelman. "Scientific workflows and clouds". In: *ACM Crossroads* 16.3 (2010), pp. 14–18.

[65] Kate Keahey, Patrick Armstrong, John Bresnahan, et al. "Infrastructure outsourcing in multi-cloud environment". In: *FederatedClouds - Workshop on Cloud services, federation, and the open cirrus summit*. ACM. 2012, pp. 33–38.

[66] Justin J Levandoski, P-A Larson, and Radu Stoica. "Identifying hot and cold data in main-memory databases". In: *ICDE - IEEE International Conference on Data Engineering*. IEEE. 2013, pp. 26–37.

[67] Guillaume Leygues. *Spotify chooses Google Cloud Platform to power data infrastructure*. 2016. URL: https://cloudplatform.googleblog.com/2016/02/Spotify-chooses-Google-Cloud-Platform-to-power-data-infrastructure.html (visited on 2017-03-14).

[68] Haoyuan Li, Ali Ghodsi, Matei Zaharia, et al. "Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks". In: *SOCC - ACM Symposium on Cloud Computing*. ACM, 2014, 6:1–6:15.

[69] *top - Linux Manpages Online*. URL: https://man.cx/top/ (visited on 2017-03-14).

[70] J. Liu, E. Pacitti, P. Valduriez, et al. "A Survey of Data-Intensive Scientific Workflow Management". In: *Journal of Grid Computing* 13.4 (2015), pp. 457–493.

[71] Ji Liu. "Multisite Management of Scientific Workflows in the Cloud". PhD Thesis. Université de Montpellier, Nov. 2016. URL: https://tel.archives-ouvertes.fr/tel-01400625/.

[72] Ji Liu, Esther Pacitti, Patrick Valduriez, et al. "Scientific workflow scheduling with provenance support in multisite cloud". In: *VECPAR - International Meeting on High-Performance Computing for Computational Science*. 2016, pp. 1–8.

[73] Ji Liu, Vítor Silva, Esther Pacitti, et al. "Scientific workflow partitioning in multisite cloud". In: *Euro-Par - European Conference on Parallel Processing*. Springer. 2014, pp. 105–116.

[74] R.J. van Looy. *SAP Business Workflow*. URL: https://wiki.scn.sap.com/wiki/display/ABAP/SAP+Business+Workflow/ (visited on 2017-03-14).

[75] Bernard Marr. *That's Data Science: Airbus Puts 10,000 Sensors in Every Single Wing!* URL: http://www.datasciencecentral.com/profiles/blogs/that-s-data-science-airbus-puts-10-000-sensors-in-every-single/ (visited on 2017-03-14).

[76] Petar Maymounkov and David Mazieres. "Kademlia: A peer-to-peer information system based on the XOR metric". In: *IPTPS - International Workshop on Peer-to-Peer Systems*. Springer. 2002, pp. 53–65.

[77] Nimrod Megiddo and Dharmendra S Modha. "ARC: A Self-Tuning, Low Overhead Replacement Cache". In: *FAST - USENIX Conference on File and Storage Technologies*. Vol. 3. 2003, pp. 115–130.

[78] Peter Mell, Tim Grance, et al. *The NIST definition of cloud computing*. NIST Special Publication 800-145. 2011. URL: http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf.

[79] *Azure Service Bus - Cloud Messaging Service*. Microsoft. URL: https://azure.microsoft.com/en-us/services/service-bus/ (visited on 2017-03-11).

[80] *HDInsight - Hadoop, Spark and R Solutions for the Cloud*. Microsoft. URL: https://azure.microsoft.com/en-us/services/hdinsight/ (visited on 2017-03-11).

[81] *Understanding Office 365 identity and Azure Active Directory*. Microsoft. URL: https://support.office.com/en-us/article/Understanding-Office-365-identity-and-Azure-Active-Directory-06a189e7-5ec6-4af2-94bf-a22ea225a7a9/ (visited on 2017-03-11).

[82] *Azure Blob Storage | Microsoft Flow*. Microsoft. URL: https://flow.microsoft.com/en-us/services/shared_azureblob/azure-blob-storage/ (visited on 2017-03-11).

[83] *Pricing - Cloud Storage*. Microsoft. URL: https://azure.microsoft.com/en-us/pricing/details/storage/blobs/ (visited on 2017-03-11).

[84] *Microsoft Azure Cloud*. Microsoft. URL: https://azure.microsoft.com/ (visited on 2017-03-11).

[85] *Azure Storage - Secure cloud storage*. Microsoft. URL: https://azure.microsoft.com/en-us/services/storage/ (visited on 2017-03-11).

[86] *Azure App Service for web, mobile and API apps*. Microsoft. URL: https://docs.microsoft.com/en-us/azure/app-service/app-service-value-prop-what-is/ (visited on 2017-03-11).

[87] *Azure Germany Cloud Computing*. Microsoft. URL: https://azure.microsoft.com/en-us/overview/clouds/germany/ (visited on 2017-03-14).

[88] *High Performance Computing | Microsoft Azure*. Microsoft. URL: https://azure.microsoft.com/en-us/solutions/high-performance-computing/ (visited on 2017-03-14).

[89] *Microsoft Azure Subscription and Service Limits, Quotas and Constraints*. Microsoft. URL: https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits/ (visited on 2017-03-11).

[90] *Automate Processes + Tasks | Microsoft Flow*. Microsoft. URL: https://flow.microsoft.com/en-us/ (visited on 2017-03-14).

[91] *ExpressRoute - Virtual Private Cloud Connections*. Microsoft. URL: https://azure.microsoft.com/en-gb/services/expressroute/ (visited on 2017-03-14).

[92] *Azure Autoscale | Microsoft Azure*. Microsoft. URL: https://azure.microsoft.com/en-us/features/autoscale/ (visited on 2017-03-11).

[93] *Azure Managed Cache Service*. Microsoft. URL: https://msdn.microsoft.com/en-us/library/azure/dn386094/ (visited on 2017-03-11).

[94] *Cloud Services - Deploy web apps and APIs*. Microsoft. URL: https://azure.microsoft.com/en-us/services/cloud-services/ (visited on 2017-03-14).

[95] Ashish Nagavaram, Gagan Agrawal, Michael A Freitas, et al. "A cloud-based dynamic workflow for mass spectrometry data analysis". In: *e-Science - IEEE International Conference on E-Science*. IEEE. 2011, pp. 47–54.

[96] *NASA Terra Project*. NASA. URL: http://terra.nasa.gov/about/ (visited on 2017-03-11).

[97] Eric H. Neilsen Jr. "The Sloan Digital Sky Survey Data Archive Server". In: *Computing in Science and Engineering* 10.1 (2008), pp. 13–17.

[98] Rajesh Nishtala, Hans Fugal, Steven Grimm, et al. "Scaling Memcache at Facebook". In: *NSDI - USENIX Symposium on Networked Systems Design and Implementation*. Vol. 13. 2013, pp. 385–398.

[99] Aisling O'Driscoll, Jurate Daugelaite, and Roy D Sleator. "'Big data', Hadoop and cloud computing in genomics". In: *Journal of biomedical informatics* 46.5 (2013), pp. 774–781.

[100] Eduardo Ogasawara, Jonas Dias, Fabio Porto, et al. "An algebraic approach for data-centric scientific workflows". In: *Proceedings of VLDB Endowment* 4.12 (2011), pp. 1328–1339.

[101] Eduardo Ogasawara, Jonas Dias, Vitor Silva, et al. "Chiron: a parallel engine for algebraic scientific workflows". In: *Concurrency and Computation: Practice and Experience* 25.16 (2013), pp. 2327–2341.

[102] Daniel de Oliveira, Kary A. C. S. Ocaña, Fernanda Baião, et al. "A Provenance-based Adaptive Scheduling Heuristic for Parallel Scientific Workflows in Clouds". In: *Journal of Grid Computing* 10.3 (2012), pp. 521–552.

[103] Daniel de Oliveira, Eduardo Ogasawara, Fernanda Baião, et al. "Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows". In: *CLOUD - IEEE International Conference on Cloud Computing*. IEEE. 2010, pp. 378–385.

[104] *Lustre*. OpenSFS. URL: http://lustre.org/ (visited on 2017-03-11).

[105] *OpenStack Open Source Cloud Computing Software*. OpenStack. URL: https://www.openstack.org/ (visited on 2017-03-11).

[106] *Software*. OpenStack. URL: https://www.openstack.org/software/project-navigator/ (visited on 2017-03-11).

[107] *Unveiling CERN cloud architecture*. OpenStack. URL: https://www.openstack.org/videos/tokyo-2015/unveiling-cern-cloud-architecture/ (visited on 2017-03-11).

[108] Philip O'Toole. *rqlite: The lightweight, distributed relational database*. URL: https://github.com/rqlite/rqlite/ (visited on 2017-03-14).

[109] Youngwoo Park, Seung-Ho Lim, Chul Lee, et al. "PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash". In: *SAC - ACM symposium on Applied computing*. ACM. 2008, pp. 1498–1503.

[110] *Pegasus Overview*. Pegasus WMS. URL: https://pegasus.isi.edu/overview/ (visited on 2017-03-14).

[111] *Creating Workflows - Pegasus WMS*. Pegasus WMS. URL: https://pegasus.isi.edu/documentation/creating_workflows.php (visited on 2017-03-14).

[112] *Metadata*. Pegasus WMS. URL: https://pegasus.isi.edu/documentation/metadata.php (visited on 2017-03-11).

[113] David Pellerin, Dougal Ballantyne, and Adam Boeglin. *An Introduction to High Performance Computing on AWS: Scalable, Cost-Effective Solutions for Engineering, Business, and Science*. AWS Whitepapers. Amazon, 2015. URL: https://aws.amazon.com/fr/whitepapers/.

[114] Andrew Pollack. *DNA Sequencing Caught in Deluge of Data*. 2011. URL: http://www.nytimes.com/2011/12/01/business/dna-sequencing-caught-in-deluge-of-data.html (visited on 2017-03-14).

[115] *PostgreSQL*. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/ (visited on 2017-03-11).

[116] Ashwin Rao, Arnaud Legout, Yeon-sup Lim, et al. "Network characteristics of video streaming traffic". In: *CoNEXT - ACM COnference on emerging Networking EXperiments and Technologies*. ACM. 2011, p. 25.

[117] Daniel A Reed and Jack Dongarra. "Exascale Computing and Big Data". In: *Communications of the ACM* 58.7 (2015), pp. 56–68.

[118] Antonio Regalado. *U.S. to Develop DNA Study of One Million People*. 2015. URL: https://www.technologyreview.com/s/534591/us-to-develop-dna-study-of-one-million-people/ (visited on 2017-03-14).

[119] Kai Ren, Qing Zheng, Swapnil Patil, et al. "IndexFS: Scaling File System Metadata Performance with Stateless Caching and Bulk Insertion". In: *SC - ACM/IEEE Conference on Supercomputing*. IEEE Press, 2014, pp. 237–248.

[120] Maria Alejandra Rodriguez and Rajkumar Buyya. "Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds". In: *IEEE Transactions on Cloud Computing* 2.2 (2014), pp. 222–235.

[121] Nick Russell, Arthur Ter Hofstede, Wil Van Der Aalst, et al. *Workflow control-flow patterns: A revised view*. Tech. rep. BPM-06-22. BPM Center, 2006, pp. 06–22.

[122] Russel Sandberg. "The Sun network file system: Design, implementation and experience". In: *Distributed Computing Systems: Concepts and Structures* (1987), pp. 300–316.

[123] Salvatore Sanfilippo. *Redis Cache*. URL: http://redis.io/ (visited on 2017-03-11).

[124] Salvatore Sanfilippo. *Who's Using Redis?* URL: https://redis.io/topics/whos-using-redis/ (visited on 2017-03-11).

[125] Frank Schmuck and Roger Haskin. "GPFS: A Shared-Disk File System for Large Computing Clusters". In: *FAST - USENIX Conference on File and Storage Technologies*. 2002.

[126] Ke Shi. "A replication and cache based distributed metadata management system for data grid". In: *SNDP - ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*. Vol. 3. IEEE. 2007, pp. 20–25.

[127] Takeshi Shibata, SungJun Choi, and Kenjiro Taura. "File-access Patterns of Data-intensive Workflow Applications and Their Implications to Distributed Filesystems". In: *HPDC - ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 746–755.

[128] Tom Simonite. *Moore's Law Is Dead. Now What?* 2016. URL: https://www.technologyreview.com/s/601441/moores-law-is-dead-now-what/ (visited on 2017-03-14).

[129] Emil Sit, Andreas Haeberlen, Frank Dabek, et al. "Proactive Replication for Data Durability". In: *IPTPS - International Workshop on Peer-to-Peer Systems*. 2006.

[130] Cooper Smith. *Facebook Users Are Uploading 350 Million New Photos Each Day*. 2013. URL: http://www.businessinsider.com/facebook-350-million-photos-each-day-2013-9/ (visited on 2017-03-14).

[131] *Swift Tutorial for Cloud and Ad hoc Resources*. The Swift Parallel Scripting Language. URL: http://swift-lang.org/tutorials/cloud/tutorial.html (visited on 2017-03-14).

[132] Alexander Thomson and Daniel J Abadi. "CalvinFS: consistent WAN replication and scalable metadata management for distributed file systems". In: *FAST - USENIX Conference on File and Storage Technologies*. 2015, pp. 1–14.

[133] *TOP500 Supercomputer Sites, November 2016*. TOP500.org. URL: https://www.top500.org/lists/2016/11/ (visited on 2017-03-11).

[134] Radu Tudoran. "High-Performance Big Data Management Across Cloud Data Centers". PhD Thesis. ENS Rennes, Dec. 2014. URL: https://tel.archives-ouvertes.fr/tel-01093767/.

[135] Radu Tudoran, Alexandru Costan, Ramin Rezai Rad, et al. "Adaptive file management for scientific workflows on the Azure cloud". In: *BigData - IEEE International Conference on Big Data*. IEEE. 2013, pp. 273–281.

[136] Radu Tudoran, Olivier Nano, Ivo Santos, et al. "Jetstream: Enabling high performance event streaming across cloud data-centers". In: *DEBS - ACM International Conference on Distributed Event-Based Systems*. ACM. 2014, pp. 23–34.

[137] *Company | About*. Twitter, Inc. URL: https://about.twitter.com/company/ (visited on 2017-03-11).

[138] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, et al. "Apache hadoop yarn: Yet another resource negotiator". In: *SOCC - ACM Annual Symposium on Cloud Computing*. ACM. 2013, p. 5.

[139] Jim Waldo. "Scaling in games and virtual worlds". In: *Communications of the ACM* 51.8 (2008), pp. 38–44.

[140] Liang Wang and Jussi Kangasharju. "Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT". In: *P2P - IEEE International Conference on Peer-to-Peer Computing*. IEEE. 2013, pp. 1–10.

[141] Lizhe Wang, Jie Tao, Rajiv Ranjan, et al. "G-Hadoop: MapReduce across distributed data centers for data-intensive computing". In: *Future Generation Computer Systems* 29.3 (2013), pp. 739–750.

[142] Janet Wiener and Nathan Bronson. *Facebook's Top Open Data Problems*. 2014. URL: https://research.fb.com/facebook-s-top-open-data-problems/ (visited on 2017-03-14).

[143] Michael Wilde, Mihael Hategan, Justin M Wozniak, et al. "Swift: A language for distributed parallel scripting". In: *Parallel Computing* 37.9 (2011), pp. 633–652.

[144] Ouri Wolfson, Sushil Jajodia, and Yixiu Huang. "An adaptive data replication algorithm". In: *ACM Transactions on Database Systems (TODS)* 22.2 (1997), pp. 255–314.

[145] Katherine Wolstencroft, Robert Haines, Donal Fellows, et al. "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud". In: *Nucleic Acids Research* 41.W1 (2013), W557–W561.

[146] Li Yu and Douglas Thain. "Resource management for elastic cloud workflows". In: *CCGRID - IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE. 2012, pp. 775–780.

[147] Qi Zhang, Ling Liu, Kisung Lee, et al. "Improving Hadoop service provisioning in a geographically distributed cloud". In: *CLOUD - IEEE International Conference on Cloud Computing*. IEEE. 2014, pp. 432–439.

[148] Paul Zikopoulos and Chris Eaton. *Understanding Big Data: Analytics for Enterprise-Class Hadoop and Streaming Data*. McGraw-Hill Osborne Media, 2011.

Part V

Appendix

Résumé en Français

Contexte

Nous vivons dans une ère où les volumes de données sont si importants et croissent si vite que l'on estime que, d'ici 2020, "*il y aura presque autant d'octets numériques qu'il y a d'étoiles dans l'univers [physique]*" (rapport annuel de la société IDC Research[1] [62]). La même étude estime que l'univers numérique atteindra 44 zettaoctets d'ici là ; Facebook a rapporté un flux entrant quotidien de 4 pétaoctets sur ses services en 2014 [142]. Ces données se présentent sous les formes les plus diverses et proviennent de sources géographiquement dispersées. L'explosion des quantités de données crée un besoin sans précédent en terme de stockage et de traitement de données, mais aussi en terme de logiciels de traitement de données capables d'exploiter au maximum ces ressources informatiques.

D'innombrables applications scientifiques et commerciales doivent traiter de (très) grandes fractions de cet univers numérique. Puisque certains jeux de données atteignant l'ordre de pétaoctets, ces applications nécessitent des environnements très performants afin de fonctionner ; elles sont traditionnellement hébergées sur des *supercalculateurs* ou sur des *clouds*. Les supercalculateurs modernes atteignent des performances de l'ordre de dizaines de pétaflops [133] ; cependant, l'accès à ces colosses de calcul est restreint, puisque l'achat et le maintien d'un supercalculateur est prohibitif pour la plupart des entreprises et des institutions. En outre, même quand un supercalculateur est disponible, seul des groupe d'utilisateurs restreints y ont accès et ce de manière prioritaire ou même réservée.

D'autre part, les clouds ont émergé comme une alternative rentable pour le calcul intensif. Dans la plupart des cas, ils se composent de plusieurs *data centers* (ou *sites*) répartis entre plusieurs zones géographiques. Ils offrent la possibilité de louer de grandes quantités de ressources fédérées pour une durée convenant à l'utilisateur et à des prix relativement abordables, tout en gardant la gestion du matériel transparente à l'utilisateur. Les clouds ont permis aux entreprises et aux universités d'augmenter leur capacité de calcul à grande échelle sans pour autant impacter leurs finances.

Ces applications à grande échelle se composent généralement de nombreux threads exécutés simultanément sur plusieurs machines. Une approche habituelle pour mettre en œuvre

[1] International Data Corporation Research.

ces applications est d'utiliser le modèle de programmation MapReduce [39], qui permet d'effectuer des calculs parallèles sur les partitions de grands ensembles de données, puis de fusionner les résultats partiels obtenus. Bien que ce modèle soit efficace pour un grand nombre de problèmes, il n'est pas adapté à de nombreux autres cas, tels que les applications avec des modèles de dépendance hétérogène, ou lorsque les données sont produites/stockées dans des lieux physiquement éloignés.

Afin de surpasser les limites de MapReduce, une pratique de plus en plus adoptée consiste à utiliser des *workflows*. Un workflow est un modèle plus abstrait pour définir formellement une application comme un graphe orienté, où les sommets désignent les tâches et les arcs les dépendances de données entre ces tâches. Grâce à ce modèle plus souple, nous sommes capables de définir plus simplement des tâches computationnelles multi-étapes. Pourtant, une question cruciale demeure : les applications à grande échelle ne sont pas seulement complexes à modéliser, mais sont aussi avides de ressources. Beaucoup de ces applications traitent des volumes de données à l'échelle de pétaoctets ; par exemple, l'expérience ALICE au CERN a besoin de stocker 1,25 gigaoctets par seconde [26], sans parler des besoins pour analyser ces données. Ainsi, dans la pratique, il devient impossible d'exécuter certaines applications dans un seul data center pour plusieurs raisons ; soit parce que les données/les calculs sont simplement trop imposants/nombreux pour les ressources disponibles, soit car les données requises sont plutôt réparties sur la planète. Par conséquent, un nouveau facteur entre en jeu : le *traitement multisite*.

À ce jour, certains fournisseurs de cloud offrent des moyens de communication entre les data centers, par exemple des bus de données [79], ou des réseaux dédiés [5], ce qui évite partiellement le surcoût dû à la configuration manuelle du réseau entre les sites. Cependant, un inconvénient plus préoccupant dans le traitement multisite est la latence des réseaux inter-sites qui est quelques ordres de grandeur plus élevée que celle d'un réseau interne de data center. Comme les workflows requièrent le transfert de grandes quantités de données d'une tâche à la suivante, les transferts au sein d'un seul site pourraient prendre un temps très court avec un système de fichiers partagé, toutefois, dans un environnement multisite, ceci pourrait se traduire par une sévère congestion du réseau et des retards, notamment avec le trafic IP qui devrait tripler dans les cinq prochaines années [34].

Il ne fait aucun doute que les clouds ont facilité les déploiements multi-site dans une certaine mesure ; néanmoins, pour faire face aux handicaps précédents et permettre l'exécution des workflows entre data centers plusieurs défis surgissent.

1. Comment minimiser l'impact des communications entre data centers ?
2. Comment équilibrer la charge de travail entre data centers afin d'éviter des goulets d'étranglement ?
3. Comment optimiser l'utilisation des ressources du cloud pour réduire les coûts ?
4. Quelles stratégies utiliser pour le stockage et transfert de données à grande échelle ?
5. Comment rassembler les tâches et les jeux de données pour minimiser des transferts ?

Alors que ces défis orientent et motivent la présente thèse, nos contributions, présentées dans la section prochaine, visent en particulier les éléments 1, 2 et 3. Les défis 4 et 5 ont été traités dans des thèses précédentes [134, 71] s'articulant au sein du même projet que la nôtre : "Z-CloudFlow : Workflows de données dans le cloud" du Centre de recherche commun Inria – Microsoft Research.

Contributions

Nos contributions de recherche sont résumées ci-dessous. Les deux premières correspondent à des travaux réalisés dans le projet Z-CloudFlow. La dernière est liée à un stage effectué au *Argonne National Laboratory* (ANL) aux États-Unis, dans le cadre du *Joint Laboratory for Extreme Scale Computing* (JLESC) et de l'équipe associée Data@Exascale.

Améliorer la performance des workflows sur le cloud grâce à la gestion adaptative de métadonnées distribuées

Lors d l'exécution typique d'une application à grande échelle, le jeu de données est partitionné en plusieurs fragments pour un traitement en parallèle. En général, pour les workflows, les données d'entrée consistent souvent en une quantité importante de fichiers de petite taille, et à mesure que le workflow s'exécute, des données intermédiaires sont générées, ce qui fait que le nombre de fichiers augmente. Dans tous les cas, de grandes quantités de métadonnées sont nécessaires pour assurer le suivi de chaque donnée (fichier, fragment, entrée d'une base de données) et de l'état général de l'exécution. Cette surcharge de métadonnées peut facilement saturer les systèmes de fichiers actuels qui sont pour la plupart conçus pour un seul data center et qui gèrent les métadonnées dans un seul serveur (voire aucun). Le scénario devient plus complexe si l'application est déployée dans un cloud multisite : chaque mise à jour de métadonnées doit être enregistrée dans un unique serveur qui est potentiellement distant et accessible via un réseau à latence élevée. Même si ces mises à jour sont effectuées en lots, il est incontestable que la performance de l'application diminuera significativement.

Afin de réduire cet impact, nous avons exploré des stratégies utilisant la sémantique des workflows dans une hiérarchie de partitionnement de métadonnées à deux niveaux combinant la distribution et la réplication. Nous avons implémenté ces stratégies pour permettre l'exécution de workflows dans un cloud multisite et nous avons validé notre approche sur quatre centres de données en utilisant des benchmarks synthétiques et des applications réelles. Par rapport à une configuration témoin centralisée, nous avons réussi à obtenir jusqu'à 28 % de gain en temps d'exécution pour une application réelle, parallèle, et géo-distribuée, et jusqu'à 50 % pour un benchmark synthétique avec une importante utilisation de métadonnées. Ce travail a été publié dans la conférence [Cluster2015].

Favoriser la propagation efficace des métadonnées fréquemment consultées dans les workflows

L'augmentation des volumes de données manipulées sur des systèmes à large échelle a conduit à une augmentation analogue de la charge liée aux métadonnées, ce qui a incité leurs concepteurs à porter une attention particulière aux gestionnaires de métadonnées. Le nombre d'opérations sur les métadonnées augmente exponentiellement avec le nombre d'éléments de données. En conséquence, une gestion des métadonnées inappropriée ou inexistante génère des goulets d'étranglement, ce qui pourrait impacter les performances du système. La gestion des métadonnées dédiée est la clé du succès dans certains systèmes de fichiers [132] ; cependant, ces solutions concernent jusqu'à présent principalement des infrastructures HPC implantées dans un seul site.

Comme nous nous intéressons par des workflows dans clouds multisite, cette contribution renforce le modèle hybride (décentralisé/distribué) de manipulation de métadonnées décrit précédemment. Nous avons analysé les métadonnées de workflows par rapport à leur fréquence d'accès et nous appelons *métadonnées chaudes* à celles qui sont plus fréquemment requises (et inversement *métadonnées froides*). Nous avons développé une approche qui permet la propagation rapide des métadonnées chaudes tout en retardant les opérations sur métadonnées froides. Cette action réduit la congestion du réseau en limitant le nombre d'opérations envoyées sur les réseaux à haute latence, améliorant ainsi le temps global d'exécution des workflows. De plus, nous avons couplé notre modèle à un moteur d'exécution de workflows afin de valider et d'adapter son applicabilité à différents scénarios scientifiques réels. Nos résultats ont révélé une amélioration systématique de plus de 20 % sur le temps d'exécution des jobs hautement parallèles des workflows, qui sont une constante dans le traitement à grande échelle. Un papier décrivant cette contribution a été accepté dans la conférence [BigData2016].

Évaluer les facteurs d'élasticité pour les *appliances* intelligentes dans des clouds privés

Une caractéristique clé du cloud est l'*élasticité*, à savoir la capacité d'approvisionner et de supprimer des ressources informatiques (généralement des machines virtuelles) en réponse à la charge de travail actuelle. En général, les nouvelles machines sont instanciées à partir d'images préconfigurées avec les logiciels et les paramètres nécessaires, ce que nous appelons *appliances*. Les workflows peuvent considérablement tirer profit de l'élasticité, puisque le nombre de machines allouées pourrait être dynamiquement ajusté selon l'usage de ressources de chaque tâche, évitant des charges inutiles dues aux temps d'inactivité. Pour atteindre cette réponse dynamique, nous devons d'abord identifier les paramètres qui augmentent/diminuent la consommation de ressources dans une application. L'identification et l'analyse de ces paramètres permettront de modéliser leur comportement pendant l'exécution, avec l'objectif ultime de prédire le moment opportun pour ajouter/enlever des ressources dynamiquement.

Dans cette contribution nous nous intéressons à un workflow d'analyse de données issues des réseaux sociaux, pour repérer des facteurs d'élasticité. Plus précisément, l'application évalue des données de Twitter pour estimer les corrélations possibles entre les déménagements (y compris professionnels) des personnes et le taux de chômage. Nous avons réalisé une analyse des différentes étapes de l'application en termes de chargement et réplication de données, le temps d'exécution et le parallélisme. Ensuite, nous avons présenté des détails sur l'ajustement de ces paramètres de configuration pour obtenir les meilleures performances. Nous avons identifié des ressources informatiques sous-utilisées, des compromis pour le partitionnement et la réplication des données, et des points de saturation des ressources pour des données correspondant jusqu'à 42 journées (~80 Go) ; tous sont des facteurs qui déclenchent le passage à l'échelle de manière élastique. Ce travail a été présenté par un poster [SC2015].

Publications

Communications en conférences internationales

[BigData2016] Luis Pineda-Morales, Ji Liu, Alexandru Costan, Esther Pacitti, Gabriel Antoniu, Patrick Valduriez and Marta Mattoso. *Managing Hot Metadata for Scientific Workflows on Multisite Clouds*. In IEEE International Conference on Big Data, Dec 2016, Washington, United States.

[Cluster2015] Luis Pineda-Morales, Alexandru Costan and Gabriel Antoniu. *Towards Multisite Metadata Management for Geographically Distributed Cloud Workflows*. In IEEE International Conference on Cluster Computing, Sep 2015, Chicago, United States.

Posters en conférences internationales

[SC2015] Luis Pineda-Morales, Balaji Subramaniam, Kate Keahey, Gabriel Antoniu, Alexandru Costan, Shaowen Wang, Anand Padmanabhan and Aiman Soliman. *Scaling Smart Appliances for Spatial Data Synthesis*. In ACM/IEEE International Conference in Supercomputing, Nov 2015, Austin, United States.

[Cluster2014] Luis Pineda-Morales, Alexandru Costan and Gabriel Antoniu. *Big Data Management for Scientific Workflows on Multi-Site Clouds*. In PhD Forum, IEEE International Conference on Cluster Computing, Sep 2014, Madrid, Spain.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Efficient Support for Data-Intensive Scientific Workflows on Geo-Distributed Clouds

Nom Prénom de l'auteur : PINEDA MORALES LUIS EDUARDO

Membres du jury :
- Monsieur LEBRE Adrien
- Monsieur SCHÖETTNER Michaël
- Monsieur ANTONIU Gabriel
- Monsieur DESPREZ Frédéric
- Monsieur SENS Pierre
- Madame PACITTI Esther
- Monsieur COSTAN Alexandru

Président du jury : Pierre SENS

Date de la soutenance : 24 Mai 2017

Reproduction de la these soutenue

Thèse pouvant être reproduite en l'état
~~Thèse pouvant être reproduite après corrections suggérées~~

Fait à Rennes, le 24 Mai 2017

Signature du président de jury

Le Directeur,

M'hamed DRISSI

Résumé

D'ici 2020, l'univers numérique atteindra 44 zettaoctets puisqu'il double tous les deux ans. Les données se présentent sous les formes les plus diverses et proviennent de sources géographiquement dispersées. L'explosion de données crée un besoin sans précédent en terme de stockage et de traitement de données, mais aussi en terme de logiciels de traitement de données capables d'exploiter au mieux ces ressources informatiques. Ces applications à grande échelle prennent souvent la forme de workflows qui aident à définir les dépendances de données entre leurs différents composants. De plus en plus de workflows scientifiques sont exécutés sur des clouds car ils constituent une alternative rentable pour le calcul intensif. Parfois, les workflows doivent être répartis sur plusieurs data centers. Soit parce qu'ils dépassent la capacité d'un site unique en raison de leurs énormes besoins de stockage et de calcul, soit car les données qu'ils traitent sont dispersées dans différents endroits. L'exécution de workflows multisite entraîne plusieurs problèmes, pour lesquels peu de solutions ont été développées : il n'existe pas de système de fichiers commun pour le transfert de données, les latences inter-sites sont élevées et la gestion centralisée devient un goulet d'étranglement.

Cette thèse présente trois contributions qui visent à réduire l'écart entre les exécutions de workflows sur un seul site ou plusieurs data centers. Tout d'abord, nous présentons plusieurs stratégies pour le soutien efficace de l'exécution des workflows sur des clouds multisite en réduisant le coût des opérations de métadonnées. Ensuite, nous expliquons comment la manipulation sélective des métadonnées, classées par fréquence d'accès, améliore la performance des workflows dans un environnement multisite. Enfin, nous examinons une approche différente pour optimiser l'exécution de workflows sur le cloud en étudiant les paramètres d'exécution pour modéliser le passage élastique à l'échelle.

Abstract

By 2020, the digital universe is expected to reach 44 zettabytes, as it is doubling every two years. Data come in the most diverse shapes and from the most geographically dispersed sources ever. The data explosion calls for applications capable of highlyscalable, distributed computation, and for infrastructures with massive storage and processing power to support them. These large-scale applications are often expressed as workflows that help defining data dependencies between their different components.

More and more scientific workflows are executed on clouds, for they are a cost-effective alternative for intensive computing. Sometimes, workflows must be executed across multiple geodistributed cloud datacenters. It is either because these workflows exceed a single site capacity due to their huge storage and computation requirements, or because the data they process is scattered in different locations. Multisite workflow execution brings about several issues, for which little support has been developed: there is no common file system for data transfer, inter-site latencies are high, and centralized management becomes a bottleneck.

This thesis consists of three contributions towards bridging the gap between single- and multisite workflow execution. First, we present several design strategies to efficiently support the execution of workflow engines across multisite clouds, by reducing the cost of metadata operations. Then, we take one step further and explain how selective handling of metadata, classified by frequency of access, improves workflows performance in a multisite environment. Finally, we look into a different approach to optimize cloud workflow execution by studying some parameters to model and steer elastic scaling.

INSA RENNES

N° d'ordre : 17ISAR 12 / D17 - 12

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coëmes • CS 70839 • F-35708 Rennes Cedex 7
Tel : 02 23 23 82 00 - Fax : 02 23 23 83 96