HAL
open science

Towards dependability and performance benchmarking for cloud computing services

Amit Sangroya

To cite this version:

Amit Sangroya. Towards dependability and performance benchmarking for cloud computing services. Networking and Internet Architecture [cs.NI]. Université de Grenoble, 2014. English. NNT : 2014GRENM016 . tel-01548465

HAL Id: tel-01548465

https://theses.hal.science/tel-01548465v1

Submitted on 27 Jun 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministériel : Arrêté n^o

Présentée par

Amit SANGROYA

Thèse dirigée par **Mme Sara Bouchenak**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Towards Dependability and Performance Benchmarking for Cloud Computing Services

Thèse soutenue publiquement le **24 Avril 2014**,
devant le jury composé de :

Mme Claudia Roncancio
Institut Polytechnique de Grenoble - Ensimag, Présidente
Mr Frédéric Desprez
INRIA, Écoles Normales Supérieures, Lyon, Rapporteur
Mr Jean-Marc Menaud
École des Mines de Nantes, Rapporteur
Mme Karama Kanoun
LAAS-CNRS, Toulouse, Examinatrice
Mme Sara Bouchenak
University of Grenoble I, Directrice de thèse

This thesis is dedicated to my parents.

For their endless love, support and encouragement

Acknowledgments

I am sincerely and heartily grateful to my advisor, Sara Bouchenak, for giving me an opportunity to work on this research topic. I appreciate the time, guidance, and emotional support, she provided me throughout my graduate studies. I feel motivated and encouraged every time I attended her meeting. I thank her for the enormous patience that she has shown towards me. I would also like to extend my gratitude to the jury members for accepting to evaluate this thesis and providing their valuable comments.

My sincere thanks also go to Damián Serrano for the useful comments, remarks and engagement through the learning process of this PhD thesis. The good advice, support and friendship of Damian, have been invaluable on both an academic and personal level, for which I am extremely grateful. I am also greatly indebted to my colleagues in the ERODS (formerly SARDES) team for the illuminating discussions they offered me, both for understanding the problem and for improving my research approaches. I want to specially thank my former group secretary, Diane Courtiol at INRIA, who was always there to help me in the administrative issues which could be otherwise difficult.

Last but not the least, I offer my regards to my family and friends, for supporting me unconditionally throughout my life.

For any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own.

Résumé

Le Cloud Computing est en plein essor, grâce à ses divers avantages, tels l'élasticité, le coût, ou encore son importante flexibilité dans le développement d'applications. Il demeure cependant des problèmes en suspens, liés aux performances, à la disponibilité, la fiabilité, ou encore la sécurité. De nombreuses études se focalisent sur la fiabilité et les performances dans les services du Cloud, qui sont les points critiques pour le client. On retrouve parmi celles-ci plusieurs thèmes émergents, allant de l'ordonnancement de tâches au placement de données et leur réplication, en passant par la tolérance aux fautes adaptative ou à la demande, et l'élaboration de nouveaux modèles de fautes.

Les outils actuels évaluant la fiabilité des services du Cloud se basent sur des paramètres simplifiés. Ils ne permettent pas d'analyser les performances ou de comparer l'efficacité des solutions proposées. Cette thèse aborde précisément ce problème en proposant un modèle d'environnement complet de test destiné à évaluer la fiabilité et les performances des services de Cloud Computing. La création d'outils de tests destinés à l'évaluation de la fiabilité et des performances des services du Cloud pose de nombreux défis, en raison de la grande quantité et de la complexité des données traitées par ce genre de services. Les trois principaux modèles de Cloud Computing sont respectivement: Infrastructure en tant que Service (IaaS), Plate-forme en tant que Service (PaaS) et Logiciel en tant que Service (SaaS).Dans le cadre de cette thèse, nous nous concentrons sur le modèle PaaS. Il permet aux systèmes d'exploitation ou aux intergiciels d'être accessibles via une connexion internet. Nous introduisons une architecture de test générique, qui sera utilisée par la suite lors de la création d'outils de tests, destinés à l'évaluation de la fiabilité et de la performance.

Les contributions de cette thèse sont les suivantes:

- Une architecture pour définir les charges de travail, de données et de fautes pour le test d'un service de Cloud. Cette architecture est destinée à la création d'outils de tests permettant l'injection de ces différents types de charges dans un service de Cloud réel, afin de produire des statistiques liées à la fiabilité, la disponibilité, ou encore la performance. La thèse démontre finalement comment l'utilisation d'une telle architecture facilite la création d'outils de tests fiables et rentables, dans le cadre des services du Cloud.

- La conception et le développement d'outils de tests destinés à l'évaluation de la fiabilité et de la performance des services MapReduce déployés dans une infrastructure de Cloud publique. L'un de ces outils est MRBS (MapRe-

duce Benchmark Suite). Il teste cinq aspects couvrant plusieurs domaines d'application, et propose un large éventail de scénarios d'exécution tels que: des applications orientées données vs. des applications orientées calcul, des applications interactives vs. des applications par lots.

- La conception et le développement d'outils de tests destinés à l'évaluation de la fiabilité et de la performance pour un service Cloud de gestion de mémoire cache distribuée. Le prototype développé se nomme MemCB (MemCached Benchmarking). Il permet l'injection de différentes fautes telles que des pannes de nœud ou des pannes de réseau dans un service Memcached en ligne. Il produira par la suite diverses statistiques liées à la performance et à la fiabilité.

Dans l'ensemble, cette thèse apporte une vision objective et systémique d'une classe émergente et importante de systèmes informatiques. Elle facilite l'adoption d'outils de tests destinés à l'évaluation de la fiabilité et de la performance, afin de mieux les quantifier et appréhender leur importance.

Mots-clés: Outils de test, Analyse comparative, Fiabilité, Tolérance aux pannes, Évaluation de performance, Cloud Computing, MapReduce, Hadoop, Memcached

Abstract

Cloud computing models are attractive because of various benefits such as scalability, cost and flexibility to develop new software applications. However, availability, reliability, performance and security challenges are still not fully addressed. Dependability is an important issue for the customers of cloud computing who want to have guarantees in terms of reliability and availability. Many studies investigated the dependability and performance of cloud services, ranging from job scheduling to data placement and replication, adaptive and on-demand fault-tolerance to new fault-tolerance models. However, the ad-hoc and overly simplified settings used to evaluate most cloud service fault-tolerance and performance improvement solutions pose significant challenges to the analysis and comparison of the effectiveness of these solutions.

This thesis precisely addresses this problem and presents a benchmarking approach for evaluating the dependability and performance of cloud services. Designing of dependability and performance benchmarks for a cloud service is a particular challenge because of high complexity and the large amount of data processed by such service. *Infrastructure as a Service (IaaS)*, *Platform as a Service (PaaS)* and *Software as a Service (SaaS)* are the three well defined models of cloud computing. In this thesis, we will focus on the PaaS model of cloud computing. PaaS model enables operating systems and middleware services to be delivered from a managed source over a network. We introduce a generic benchmarking architecture which is further used to build dependability and performance benchmarks for PaaS model of cloud services.

Specifically, the dissertation contribute the following:

- A reusable architecture to define workload, dataload and faultloads for benchmarking a cloud service. This architecture is further used to build benchmarks that allows to inject various workloads, dataloads and faultloads, in a real cloud service and produce extensive reliability, availability and performance statistics. Finally, the dissertation demonstrates how the use of a general architecture benefits to build dependability and performance benchmarks for cloud services in a cost efficient manner.

- Design and development of dependability and performance benchmark for MapReduce service deployed in a public cloud infrastructure. One interesting outcome is MRBS (MapReduce Benchmark Suite), a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems. MRBS includes five benchmarks covering several application domains and a wide range of execution scenarios such

as data-intensive vs. compute-intensive applications, or batch applications vs. online interactive applications.

- Design and development of dependability and performance benchmark for a distributed memory object caching cloud service, where the software prototype developed is named MemCB (MemCached Benchmarking). This supports injection of various faults such as node faults and network faults in an online Memcached service and produce the performance and dependability statistics.

Overall, the dissertation develops a more objective and systematic understanding of an emerging and important class of computer systems. The work in this dissertation helps further accelerate the adoption of dependability and performance benchmarks to get better understanding of prominent quality attributes.

Keywords: Benchmarking, Dependability, Fault tolerance, Performance, Cloud Computing, MapReduce, Hadoop, Memcached

Contents

1 Introduction **1**

1.1 Background & Motivation 3

1.2 Problem Statement & Research Challenges 6

1.3 Contributions of the Thesis 9

1.4 Main Results 10

1.5 Organization of the Document 12

2 Related Work **15**

2.1 Introduction 17

2.2 Existing Approaches of Dependability Benchmarking 19

2.3 Existing Approaches of Performance Benchmarking 22

2.4 Discussion 30

3 Towards a General Architecture for Dependability & Performance Benchmarking **35**

3.1 Introduction 37

3.2 Objectives & Approach 37

3.3 General Definitions & System Model 41

3.3.1 Workload 42

3.3.2 Faultload 42

3.3.3 Dataload 43

3.3.4 Dependability & Performance Analysis 43

3.4 Design Principles 45

3.4.1 Architecture Overview 45

3.4.2 General Software Framework 46

3.5 On the Generality of the Proposed Architecture 49

3.5.1 Reduced development costs 51

3.5.2 Better usability 52

3.5.3 Higher adaptability 52

3.6 Summary 52

4 MRBS: Dependability & Performance Benchmarking for MapReduce 55
4.1 Introduction 58
4.2 Background 60
4.2.1 MapReduce 60
4.2.2 Hadoop MapReduce 60
4.2.3 Fault Tolerance in Hadoop 61
4.3 Overview of MRBS 62
4.3.1 MRBS Benchmark Suite 66
4.3.2 MRBS Workload 68
4.3.3 MRBS Dataload 69
4.3.4 MRBS Faultload 71
4.3.5 MRBS Faultload Builder 72
4.3.6 MRBS Fault Injection 72
4.3.7 Performance and Dependability Analysis in MRBS . . 74
4.4 On the Portability of MRBS Software 76
4.4.1 Portability of Fault Builder 76
4.4.2 Portability of Fault Injection 77
4.4.3 Portability of Performance and Dependability Analysis 77
4.4.4 Portability of Workload and Dataload Injection 77
4.4.5 Portability of MRBS Experiment Deployer 77
4.5 Experimental Evaluation 78
4.5.1 Experimental Setup 78
4.5.2 Performance Evaluation Under Workloads 78
4.5.3 Dependability Evaluation Under Faultloads 81
4.6 Summary 83

5 Use Cases 87
5.1 Introduction 89
5.2 Scalability With Regard To Cluster Size 89
5.3 Scalability With Regard To Data Size 91
5.4 How Many Faults Are Tolerated 91
5.5 Comparing Dependability of MapReduce Frameworks 93
5.6 Comparing Performance of MapReduce Frameworks 94
5.7 Summary 97

6 MemCB: Dependability & Performance Benchmarking for Memcached 99
6.1 Introduction 101
6.2 Background 102
6.2.1 Memcached 102
6.2.2 Fault Tolerance in Memcached 103
6.3 Overview of MemCB 104
6.3.1 MemCB Workload 104
6.3.2 MemCB Dataload 105
6.3.3 MemCB Faultload 105
6.3.4 MemCB Fault Injection 105
6.3.5 Dependability and Performance Analysis in MemCB . 106
6.4 Experimental Evaluation 107
6.4.1 Experimental Setup 107
6.4.2 Experimental Results 107
6.5 Summary 109

7 Conclusions and Perspectives 111
7.1 Conclusions 113
7.2 Perspectives 115

A Annexes 117
A.1 Properties file of MRBS 119

Bibliography 121

Chapter 1

Introduction

Contents

1.1 Background & Motivation 3
1.2 Problem Statement & Research Challenges 6
1.3 Contributions of the Thesis 9
1.4 Main Results 10
1.5 Organization of the Document 12

1.1 Background & Motivation

Cloud computing services emerged as a result of work towards organizing and provisioning computational resources. Typically, these services are owned by service providers that let consumers utilize cloud resources in a pay-as-you-go fashion: the consumer pays only for the resources that were actually used to solve its problem (for example: bandwidth, storage space, CPU utilization). This phenomenon not only increased revenue for cloud service providers, but also decreased costs for cloud service users.

Cloud computing promises to provide reliable and user-friendly services delivered through next-generation data centres that are build on virtualized computational and storage technologies. As cloud computing is evolving, many cloud services are provided. By and large, these services are being used to manage large amounts of varying and complex data[1]. MapReduce is one of the example of a popular Big Data cloud service, largely used by companies for a wide range of applications such as log analysis, data mining, scientific computing, bioinformatics, decision support or business intelligence. MapReduce and other such cloud computing services are increasingly being used in various industry domains such as financial services, retail, entertainment, or healthcare.

Use of cloud services for large scale data analysis has a series of interesting advantages:

- **Performance and scalability** are the advantages that can be achieved by migrating the data and computation to a cloud service. Since, it is in the interest of the providers to serve as many customers as possible, clouds can easily grow to huge sizes. Thus, a consumer is able to utilize virtually an unlimited number of resources, provided it has the money to pay for them.

- **Reliability and availability** are other potential benefits that can be guaranteed using a cloud service since such service is supported upon a reliable distributed storage system. Cost and reliability are one of the main drivers for the interest in cloud computing. Service availability is among the top challenges and opportunities for cloud service providers [Armbrust et al., 2010]. *Google Search* is known for being highly available, to the point that even a small disruption becomes a news. Users expect similar availability from new cloud services.

- **Data can be better managed**. Many of the big data and business intelligence tools provide service users with the freedom and flexibility

[1] This data is also referred as *big data*.

to work with data without going through too many complex technical details.

Cloud computing deployment model can be among any of the following types [Rimal et al., 2009]:

- **Public Cloud**

  This is the traditional mode of cloud computing, where an outside service provider is responsible for providing and dynamic provisioning of data and computation resources over the web. Technically there is no difference between public and private cloud architecture, however, security consideration may be substantially different for services (applications, storage, and other resources) that are made available by a service provider for a public audience and when communication is effected over a non-trusted network. Generally, public cloud service providers like Amazon AWS, Microsoft and Google own and operate the infrastructure and offer access only via Internet.

- **Private Cloud**

  In this mode, data and processes are managed within the organization without the restrictions of network bandwidth, security and other legal issues that public clouds have. The initial investment needed to build a private cloud is generally high. However, this ensures better privacy and data security than a public cloud service.

- **Hybrid Cloud**

  This mode consists of multiple internal and external service providers. It can be considered as a composition of two or more clouds (public or private). Using this model, service providers can obtain higher degrees of fault tolerance and usability without dependency on internet connectivity. Hybrid cloud architecture requires both on-premises resources and off-site (remote) server-based cloud infrastructure.

- **Community Cloud**

  In this mode of cloud computing, several organizations share and support a cloud infrastructure. The goal of a community cloud is to have participating organizations realize the benefits of a public cloud such as multi-tenancy and a pay-as-you-go billing structure, with the added level of privacy, security and policy compliance usually associated with a private cloud. The community cloud can be either on-premises or off-premises, and can be governed by the participating organizations or by a third-party service provider.

Using any of these modes, cloud computing services can be primarily classified into three layers, as illustrated in Figure 1.1.

Infrastructure as a Service (IaaS). IaaS model enables the entire infrastructure to be delivered as a service over a network, including storage, routers, virtual systems, hardware and servers. Rather than directly renting servers, software, disks or networking equipment, cloud consumers customize virtual machine images, store the images and application data remotely using the storage service, and then finally launch multiple VM instances on the cloud. Fees are charged on an utility basis that reflects the amount of raw resources used: storage space-hour, bandwidth, memory consumed, etc. A popular cloud service provider is Amazon, which provides its service by Amazon EC2 [Ama, 2011a]. Other examples of IaaS providers are Windows Azure, Rackspace, and IBM smartcloud [Sma, 2013], [Rac, 2013].

Platform as a Service (PaaS). PaaS model enables operating systems and middleware services to be delivered from a managed source over a network. It provides higher level programming and execution environments. Services at this level aim at freeing the consumer from having to configure and manage industry-standard application frameworks (for example Hadoop [Had, 2011a]), on top of which distributed applications are build, directly at IaaS level. Google App Engine is a well known PaaS provider [Goo, 2011].

Software as a Service (SaaS). At the highest level is SaaS, which aims at delivering end user applications directly as a service over the Internet, freeing the consumer from having to install any software on his/her local machine. Most often, a simple web browser is enough to perform all necessary interactions with the application. Google Apps is one of the leading examples of a SaaS provider.

As cloud services emerged, traditional applications are also being migrated to the cloud environments because of scalability objectives. There have been performance issues related to migration of traditional applications to a cloud environments. This is because applications not optimized for cloud-based platforms. Sometimes, there might be some restrictions related to data placements because of security challenges that further aggregates the performance challenges. For some critical applications, performance is not enough. Cloud service customers also need to guarantee a set of application-specific availability levels. Measuring and analysing the performance and dependability of cloud services is a serious challenge. State of art approaches for dependability and performance benchmarking suffers from cloud computing services specific design challenges. Cloud computing is a relatively new domain. There are very few standard dependability and performance benchmarks designed for cloud computing services.

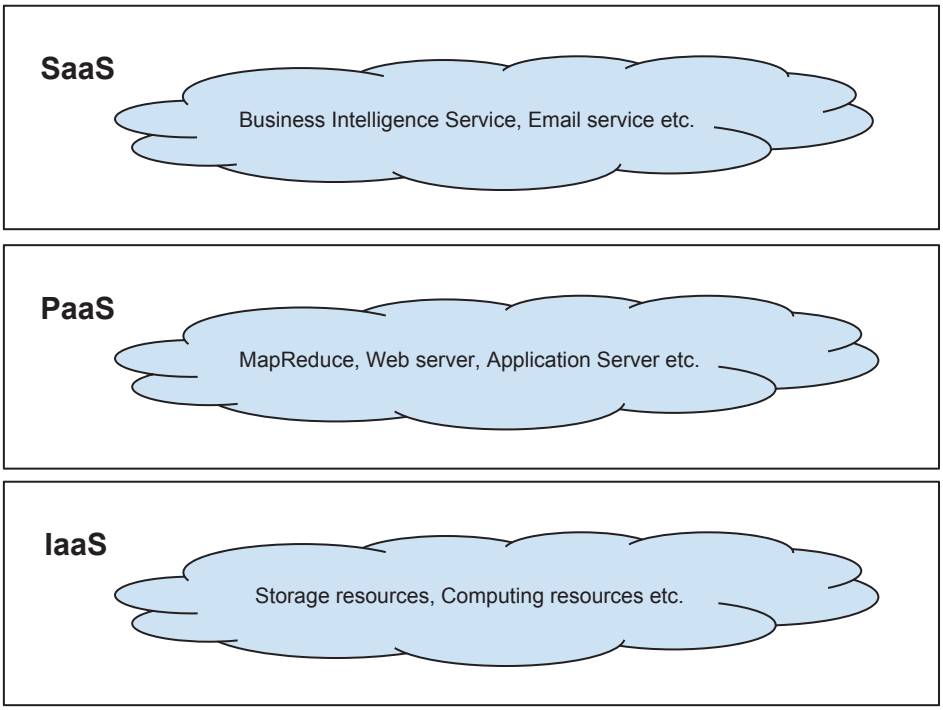


Figure 1.1: Cloud computing services and technologies as a stack of layers

1.2 Problem Statement & Research Challenges

Cloud computing models seems to be attractive because of various benefits such as: scalability, cost and flexibility to develop new software applications. However, there are still various challenges related to availability, reliability, performance and data security issues of cloud services. Table 1.1 shows some recent incidents, where well known cloud services remains unavailable (or took high time to load) for a significant time. This data has been taken from *NetworkWorld*[2] [Net, 2013]. The reasons for unavailability of these cloud services were not very clear. For example, In the case of unavailability issue of Amazon, there were speculations of denial-of-service attacks. However, experts believed that it might be due to internal issues in the cloud service. Industry experts also estimated that Amazon could have potentially suffered close to 5 million dollars of revenue for that single hour of offline time. Interestingly, the reasons of unavailability for other cloud services were minor issues such as an outdated network control software, faulty hardware, DNS problem or an internal router problem. There are various other examples of unavailability of cloud services in the recent past [CRN, 2012, Inf, 2011]. These incidents

[2]Network World is a weekly IT publication that provides news and information to network executives.

Table 1.1: Incidents of cloud outages in 2013 [Net, 2013]

Name of the Company/Service	Outrage Date	Duration of the Outrage	Outrage Issue
Dropbox	January 10, 2013	16 hours	Service Unavailable
Facebook	January 28, 2013	2-3 hours	Service Unavailable
Amazon	January 31, 2013	49 minutes	Service Unavailable
Microsoft/Outlook.com	February 1, 2013	2 hours	Service Unavailable
Google Drive	March 18-19, 2013	17 hours	Slow Load time
Dropbox	May 30, 2013	90 minutes	Service Unavailable

reveal the fact that there is a need to address dependability issues for easy and faster adoption of cloud services by the end users.

Dependability is an important issue for the customers of cloud computing who want to have guarantees in terms of reliability and availability. Although much work has been done in security and hardware in general distributed computing systems, many problems still lie in characterization of dependability of cloud computing. There is a lack of efficient approaches to characterize the availability, reliability and performance of cloud services. We argue that traditional approaches to characterize dependability and performance, may not be appropriate for modern applications in cloud computing. In this context, there has been a considerable interest in improving the dependability and performance of cloud services.

Many studies investigated the dependability and performance of cloud services, ranging from job scheduling to data placement and replication, migration of virtual machines, adaptive and on-demand fault-tolerance to new fault-tolerance models [Voorsluys et al., 2009, Jackson et al., 2010, Pham et al., 2012]. However, the ad-hoc and overly simplified settings used to evaluate cloud service dependability and performance improvement solutions, pose significant challenges to the analysis and comparison of the effectiveness of these solutions [Stantchev, 2009]. Performance of cloud services is evaluated using inadequate workloads, which may not represent real-world scenarios. Approaches to characterize and empirically evaluate dependability and performance of cloud services using realistic workloads, dataloads and faultloads, are missing.

This thesis precisely addresses this issue and presents a benchmarking approach for evaluating the dependability and performance of cloud services. Our objective in this dissertation is to create the process of dependability and performance benchmarking for cloud services more objective and systematic. We first introduce a benchmarking architecture which is further used to build dependability and performance benchmarks for PaaS model of cloud services. The work in this dissertation further helps in accelerating the adoption of

dependability and performance benchmarks and therefore aid in getting a better understanding of prominent quality attributes such as performance and dependability.

Designing of dependability and performance benchmark for a cloud service is a particular challenge because of high complexity and the large amount of data processed by such service. Few reasons, why achieving dependability in addition to performance is a serious challenge for cloud services are following.

- **Heterogeneity of cloud infrastructures.** Different technologies from different competing vendors of cloud infrastructure need to be integrated for establishing a reliable system. This integration makes the cloud services complex and creates multiple centres for failures. Sharing of cloud resources by entities that engage in a wide range of behaviors, can expose cloud applications to increased risk levels.

- **Challenges of public cloud infrastructures.** Using a public cloud means that the platform is no longer under one's own control. Depending on the type of the cloud service, the platform may mean one of several things; at least it includes the physical machines but may also include higher layers like a complete runtime system which introduces new challenges. Most importantly, the underlying physical resources (i.e., the machines and the network) are shared between multiple customers and applications and cannot be actively managed.

- **Multiple cloud platform services.** Multiple instances of an application might be running on several virtual machines. In the event of server failures, there are automatic fail-over mechanisms that guarantee against data loss. However, there are no techniques to measure the effectiveness of these fail-over mechanisms for cloud services.

- **Dependability and performance issues in cloud services.** Availability, reliability, and performance are serious challenges for applications hosted on cloud infrastructures. There are a number of areas that impact the dependability and performance such as network vulnerability, multi-site redundancy and storage failure. Without efficient approaches that can quantify dependability and performance levels, cloud service providers cannot provide a minimum service level agreement (SLA) guarantee.

- **Quantifying the dependability and performance in cloud services.** The quality attributes such as reliability, availability and performance are very critical for cloud computing services. But, it is hard

to analyze these attributes due to the complex characteristics of cloud services such as massive-scale data sharing, wide-area network, heterogeneous software/hardware components and complicated interactions among them. Moreover, there are no formal metrics for evaluating reliability, availability and performance for cloud computing services.

Design of a performance and dependability benchmark for a cloud computing service must address these challenges. The benchmark must cover these scenarios in its implementation to provide a better realistic estimate of performance and dependability levels of the service to the users. The benchmark must provide a variety of execution scenarios that are realistic for cloud services. In addition, it should also provide a range of fault scenarios that can be emulated by the users. This thesis is an attempt to make cloud services dependability and performance benchmarking process more focused and systematic.

1.3 Contributions of the Thesis

In this thesis, we consider the problem of benchmarking dependability and performance of cloud services. Specifically, this dissertation contributes the following:

- A reusable architecture to define workload, dataload and faultloads for benchmarking cloud services. This architecture is further used to build benchmarks that allows to inject various workloads, dataloads and faultloads, in a real cloud service and produce extensive reliability, availability and performance statistics. Finally, the dissertation demonstrates how the use of a general architecture benefits to build dependability benchmarks for cloud services in a cost efficient manner.

- Design and development of dependability and performance benchmark for MapReduce service deployed in a public and private cloud infrastructures. One interesting outcome is MRBS, a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems. MRBS includes five benchmarks covering several application domains and a wide range of execution scenarios such as data-intensive vs. compute-intensive applications, or batch applications vs. online interactive applications.

- Design and development of dependability and performance benchmark for a distributed memory object caching cloud service, where the soft-

ware prototype developed for dependability and performance benchmarking is named as MemCB. This benchmark supports injection of various faults such as node faults and network faults in an online Memcached service and produce the performance and dependability statistics.

Overall, the dissertation develops a more objective and systematic understanding of an emerging and important class of computer systems. The work in this dissertation further helps in accelerating the adoption of dependability and performance benchmarks to get better understanding of prominent quality attributes such as performance and dependability.

1.4 Main Results

Publications

The work in this thesis led to following major publications.

- A. Sangroya, D. Serrano and S. Bouchenak. **Experience with Benchmarking Dependability and Performance of MapReduce Systems**. *IEEE Transactions on Dependable and Secure Computing (TDSC), under submission*
- A. Sangroya, D. Serrano, S. Bouchenak. **Benchmarking Dependability of MapReduce Systems**. *31st IEEE International Symposium on Reliable Distributed Systems (SRDS). Irvine, California, Oct 2012*
- A. Sangroya, D. Serrano and S. Bouchenak. **MRBS: Towards Dependability Benchmarking for Hadoop MapReduce**. *Workshop on Big Data Management in Clouds (BDMC) in conjunction with Euro-Par. Rhodes Island, Greece, Aug. 2012*
- A. Sangroya, D. Serrano, S. Bouchenak. **Towards a Dependability Benchmark Suite for MapReduce**. *Poster at EuroSys 2012, Bern, Switzerland. Apr 2012*
- L. Lemke, A. Sangroya, D. Serrano and S. Bouchenak. **Evaluer la tolérance aux fautes de systèmes MapReduce**. *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS). Grenoble, France, Jan 2013*

Software Developed

Two software prototypes are developed as part of this thesis: MRBS and MemCB. The distribution of the prototypes developed as part of this

thesis is free and publicly available. MRBS is available as a software prototype to help researchers and practitioners to better analyze and evaluate the dependability and performance of MapReduce systems; it can be downloaded from: http://sardes.inrialpes.fr/research/mrbs.

Table 1.2: Statistics from MRBS website

Visits	980
Unique Visitors	550
Pageviews	3,394
Average Visit Duration	00:02:28
% New Visits	55.71%
Country	France, United States, Brazil, China, India, United Arab Emirates, Germany, Spain, Portugal, Canada

Table 1.2 shows the statistics from the MRBS website. Between June 5, 2012 and January 1, 2014, MRBS webpage is visited 980 times (including 550 unique visitors). The visitors comprises of 410 visitors from France, 160 from United States, 109 from Brazil, and others from China, India, United Arab Emirates, Germany, Spain, Portugal and Canada. MRBS software prototype is downloaded and being used by approximately 60 academic and research groups. People from industry and academia download MRBS for various reasons such as:

- For benchmarking an in-house developed MapReduce framework.
- Comparing two different versions of Hadoop framework.
- Evaluating Hadoop performance.
- Tuning the Hadoop for best values of configuration parameters.
- Testing fault-tolerance of large scale systems including MapReduce systems.
- Finding out the performance bottlenecks of Hadoop MapReduce.
- Studying the performance and scalability of MapReduce applications in local cluster.
- Looking for an alternative to micro-bencmarks such as TeraSort for Hadoop clusters.

1.5 Organization of the Document

This thesis is organized as follows:

Chapter 1

In Chapter 1, we provided an introduction to the problem of dependability and performance benchmarking of cloud services. We also presented the addressed scientific and technical challenges, and a summary of the main contributions of this thesis.

Chapter 2

Chapter 2 discusses the existing work related to the problem of dependability and performance benchmarking of cloud services. This chapter presents the state of art of dependability and performance benchmarking. We focus upon the work done in the area of dependability and performance benchmarking of cloud services. However, since cloud computing is relatively a new paradigm, we also study the approaches in other domains of computing such as web servers, hardware, database systems, middleware etc. The generic dependability and performance benchmarking architecture proposed in this thesis is based on the ideas and principles proposed in designing benchmarks in existing application domains. Moreover, in this chapter, we also present the scientific and research challenges to design and develop dependability and performance benchmark for cloud computing services.

Chapter 3

In Chapter 3, we introduce the generic architecture of dependability and performance benchmarking of cloud services. We validate the architecture by performing two case studies, which are discussed in detail in Chapters 4 and 6. This chapter consists of two main parts. The first part presents the system model including the general terms and definitions used in designing the benchmark architecture. We define terms such as workload, faultload and dataload in the context of cloud computing environments. Thereafter, we present the generic architecture and the details of how such architecture can be used to build software frameworks. The second part is a summary of the case studies that are done to validate the proposed architecture. The detailed case studies are presented in the following chapters. We conclude this chapter by highlighting the advantages of this architecture. Here, we describe how the use of a generic architecture can help to reduce the cost in building new software prototypes increasing further the usability and adaptability.

Chapter 4

MapReduce is a widely used cloud service for large data processing. Hadoop is a popular framework and run time environment that support MapReduce service deployed on private and public cloud. Hadoop supports fault tolerance

by automatic management of data and computation in case of software and hardware faults. This chapter, thus begins with providing a rich background on MapReduce framework. It then present a background on hadoop and how it supports fault tolerance. It then presents MRBS a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems Thereafter, details of using MRBS for dependability and performance and analysis are presented. This chapter also includes a wide experimental evaluation with MRBS.

Chapter 5

MRBS has several possible uses, among which helping developers and testers to better analyze the fault-tolerance of MapReduce systems, or to better choose the configuration of the MapReduce cluster to provide service level guarantees. This chapter includes interesting case studies with using MRBS. The case studies are performed both from the perspective of a cloud service provider and cloud service user.

Chapter 6

To validate the generality of architecture proposed in Chapter 3, we make use of the same design to build a prototype to another cloud service Memcached. This chapter describes the design and development of this prototype. It shows, first of all, an example Memcached service, fault tolerance in Memcached service. Thereafter, this chapter provides an overview of faultload, workload and dataload for this benchmark. Then it covers the experiments done to benchmark dependability and performance of Memcached service.

Chapter 7

This chapter summarizes the main contribution of this work. It also opens new elements of thought and some research perspectives.

Chapter 2

Related Work

Contents

2.1 Introduction 17
2.2 Existing Approaches of Dependability Benchmarking 19
2.3 Existing Approaches of Performance Benchmarking 22
2.4 Discussion 30

2.1 Introduction

Performance and dependability benchmarking allows to evaluate the impact of faults over the various quality aspects of software products, ranging from operating systems, to databases and web servers. The basic idea is to define an evaluation process that proposes a set of performance and dependability measures to characterise the quality of a particular software component; identifies the execution profile and experimentation setup to deploy; establishes the experimental procedure to follow; the time available for experimentation; and the procedure to retrieve the measurements required to deduce the performance and dependability measures from the system under test. Fault injection is a well known basic technique underpinning this type of benchmarking. Faults are used to emulate during experimentation the occurrence of disturbances and observe the resulting system reaction.

Benchmarking is particularly a very important technique for evaluating distributed systems and services. It has been invaluable in helping service providers and users identify problems, analyze causes, and evaluate solutions. Various research and industry standard performance benchmarking solutions exist for some application domains such as TPC-C that evaluates on-line transaction processing (OLTP) systems [TPC, 2011b], TPC-H for benchmarking decision support systems [TPC, 2011a].

The economy model behind cloud computing services prompted their adoption especially in the private sector. Industry giants such as: Amazon, Google, Microsoft, etc. develop and offer a wide range of cloud services. At the same time, cloud projects are also under development in academia as a series of research projects and open source initiatives [Cerbelaud et al., 2009]. Following is the list of popular cloud computing service providers:

Amazon EC2. provides a virtual computing environment that exposes a web service interface to the consumer through which it can launch virtual instances of a variety of operating systems, that can be loaded with custom application environments [Ama, 2011a]. The consumers can dynamically adjust the number of such instances through the same interface. A large pool of predefined virtual machine images, called Amazon Machine Images (AMIs) is provided, that can be directly used as such or customized to form new AMIs. The cost for using EC2 is measured in instance-hours. A specialized storage service, Amazon S3, is provided that is responsible to store both AMIs and consumer data [Ama, 2013].

Google App Engine. is a PaaS that enables consumers to to build and host web applications [Goo, 2011]. It offers fast development and deployment that is coordinated through simple, centralized administration. The service is

free up to a certain resource utilization level, after which a low pricing scheme is applied. Fee is charged for storage space-hour, bandwidth and CPU cycles required by the application.

Microsoft Azure. is the cloud service from Microsoft that runs on a large number of machines, all located in Microsoft data centres [Win, 2013]. It is based on a fabric layer that aggregates the computational resources into a whole, which is the used to build compute and storage services that are offered to the consumer. Developers can build applications on top of languages commonly supported by Windows, such as C#, Visual Basic, C++, Java, ASP.NET, using Visual Studio or another development tool.

Nimbus. is an open source toolkit that allows institutions to turn their cluster into an IaaS cloud [Nim, 2013]. This is interface-wise compatibility with the Amazon EC2 API. Data storage support is provided by Cumulus, which is compatible with the Amazon S3 API. It is based on an extensible architecture that allows easy customization.

Eucalyptus. is an open-source toolkit that started as a research project at University of California, Santa Barbara [Nurmi et al., 2009]. It implements IaaS using existing Linux-based infrastructure found in modern data centres. Its interface is compatible with Amazon's EC2 API enabling movement of workloads between EC2 and data centres without modifying any code.

OpenNebula. is another open-source toolkit, specifically designed to support building clouds [Sempolinski and Thain, 2010]. It can be integrated with a wide range of storage and networking solutions to fit a broad class of data centres, in order to form a flexible virtual infrastructure which dynamically adapts to changing workloads.

Despite of extensive and growing demand of cloud computing services, little has been reported on dependability and performance benchmarking in cloud computing services. This motivates and facilitates the advances proposed in the dissertation. This chapter gives a quick overview of dependability and performance benchmarking in various application domains. We focus upon the dependability and performance benchmarks proposed in cloud computing services such as MapReduce, studied in this dissertation. This chapter summarises the results of a wide survey of existing approaches for performance and distributed computing in various diverse areas of computing systems such as distributed systems, hardware, operating systems, database systems, web services, cloud computing among others.

In particular, we categorize the existing work into two dimensions:

1. Benchmark approaches focusing upon the performance issues.

2. Benchmark approaches focusing upon the dependability issues.

2.2 Existing Approaches of Dependability Benchmarking

In this section, we present a brief summary of dependability benchmarks proposed in various application domains. We conduct a small study in various application domains of computing to see if dependability benchmarks exist in those domains or not. The application domains that we consider are hardware; cluster computing; operating systems; database systems; web servers; and web services. Here, we present a brief summary of state of art approaches in dependability benchmarking in all these domains:

Hardware

In the domain of *hardware* dependability benchmarking, *Brown et al.* proposed a methodology to measure the availability of the software RAID systems [Brown and Patterson, 2000]. They propose two different kinds of fault workloads: Single-fault workloads and Multi-fault workloads. A single-fault workload consists of just a single fault, whereas multi-fault workloads consist of a series of faults. Some examples of their faultloads are hardware errors or mechanical errors; parity errors; power failures; and hard disk errors. Their experimental evaluation consist of fault injection either in transient mode or in sticky mode. In transient mode, faults appeared once and then disappeared. However, in sticky mode, faults continued to manifest themselves after the first injection. The dependability metrics that are used in their evaluation is availability.

Ruiz et al. proposed a dependability benchmarking methodology for comparing Commercial-Off-The-Shelf (COTS) components in software development on the basis of dependability [Friginal et al., 2011]. Their benchmarking proposal to evaluate any hardware system consists of a thorough fault injection approach consisting of malicious attacks along with measures related to reliability, security and performance.

Operating Systems

Kanoun et al. presented the specification of a dependability benchmark for operating systems [Kanoun et al., 2005]. This work is part of a big European project on dependability benchmarking, named as **DBench**. Authors provided software prototypes for qualitatively and quantitatively evaluating two families of popular operating systems i.e. Windows and Linux. Their faultload primarily comprises of corrupt system calls using bit-flip techniques. Dependability measures in their benchmark are: operating system robustness, reaction time and restart time in the presence and absence of faults.

Koopman et al. proposed benchmarks to test robustness of operating sys-

tems by feeding corrupt data to system calls [Koopman et al., 1997]. They compared five operating systems by analysing robustness which is measured as the ability of a system to cope with errors. In this work, authors do not explicitly injected a faultload, but rather assign a representative workload. For example, they injected a workload that assigns different tasks to the operating system such as read and write to a file. Finally they observe the system behaviour by calculating the total number of crashes, aborts and restarts.

Barbosa et al. described a dependability benchmark to evaluate partitioning operating systems [Barbosa et al., 2011]. In contrast to traditional process based operating system such as Linux, in a partitioning operating system, memory (and possibly CPU time as well) is divided among statically allocated partitions in a fixed manner. They included both hardware and software faultloads e.g. single bit-flips into processor registers and memory. Interestingly, they also measured the coverage of the injected faults. Finally, they used low level metrics e.g. arithmetic mean of register errors and memory errors to compute the dependability levels. One of the limitations of this benchmark is that it is proposed for a very specific system only i.e. µ*C/OS-II with Screen* system. Thus, it is not portable to other application domains. Moreover, they do not presented a detailed architecture that can be extended by others.

Database Systems

In the domain of *database systems*, *Vieira et al.* proposed a dependability benchmark for OLTP systems using the workload of the TPC-C performance benchmark [Vieira and Madeira, 2003]. They evaluated both the performance and key dependability features of OLTP systems, with emphasis on availability. Their faultload consisted of operator faults, software faults, and hardware faults. In addition to basic setup, the workload, and the performance measures specified in the *TPC-C* performance benchmark, they added two new measures related to dependability. One key feature of their dependability benchmark is that their faultload is portable across typical OLTP systems.

Web Servers

Duraes et al. presented a benchmark for evaluating the the dependability of web servers [Durães et al., 2004]. They performed a case study involving two widely used web servers (Apache and Abyss) running on top of three different operating systems. They used the *SPECWeb99* benchmark as starting point, adopting the workload and performance measures from this performance benchmark, and added the faultload and new measures related to dependability. Their faultload consisted of software faults, hardware faults, operator faults and network faults. Software faults were emulated by reproducing directly at low-level code the processor instruction sequences that

represent programming errors. Hardware and operator faults were emulated using network failures, web server shutdowns and abrupt server reboots. This benchmark can be used to compare several web servers and decide which one is best suited to include in a larger information system.

Web Services

Web Services are a key technology in Service Oriented Architecture (SOA) environments, which are increasingly being used in business critical applications. A web service is a software component that exposes a given functionality that can be assessed by service consumers in an interoperable manner. *Vieira et al.* proposed a benchmarking approach for the evaluation of the robustness of web services [Vieira et al., 2007]. They made use of standard *TPC-App* performance benchmark with two different implementations of the web services. Two options provided for workload generation in their benchmark were: (1) user defined workload and (2) random workload. Their faultload was composed of errors such as invalid web services call parameters that were applied in order to discover both programming and design errors. The benchmark tool included a fault injection component that acted like a proxy that intercepted all client requests (generated by the workload emulator component); performed a set of mutations in the Simple Object Access Protocol (SOAP) message; and forwarded the modified message to the server. They illustrated the proposed approach by evaluating several publicly available web services.

Laranjeiro et al. [Laranjeiro et al., 2008] presented a dependability benchmark tool named **wsrbench**, for measuring robustness of web services. This tool is available online at `http://wsrbench.dei.uc.pt` and requires very little configuration effort to use. It is also provided with a web based interface that allows users to perform configurations and visualize the results of tests. They demonstrated the effectiveness by testing 100 publicly available web services. Like the previous work, they also provided two options for workload generation: (1) user defined workload and (2) random workload [Vieira et al., 2007]. Their faultload consisted of several faulty client requests including software faults such as null strings, empty strings and alphanumeric strings. Such tools can be used by service vendors in evaluating and improving the robustness of their web services implementations before deployment.

MapReduce Systems Works that have been devoted to the study and improvement of fault-tolerance and performance of MapReduce, motivate the need of MapReduce dependability and performance benchmarking. These works include adaptive fault-tolerance [Jin et al., 2012, Lin et al., 2010], on-demand fault-tolerance [Fadika and Govindaraju, 2010] and extending MapReduce with other fault-tolerance models [Bessani et al., 2010, Ko et al., 2010],

task scheduling policies in MapReduce [Isard et al., 2009, Zaharia et al., 2010], cost-based optimization techniques [Herodotou and Babu, 2011], and replication and partitioning policies [Ananthanarayanan et al., 2011, Eltabakh et al., 2011].

These works are usually evaluated in an ad-hoc way, using microbenchmarks such as MapReduce *sort*, *grep* and *word count* programs introduced in [Dean and Ghemawat, 2004]. Although some low-level tools exist to test fault-tolerance of Hadoop MapReduce and HDFS [Had, 2011b], there is no principled way to describe faultloads, and to measure reliability and availability of MapReduce clusters. To the best of our knowledge, MRBS is the first dependability and performance benchmark suite for MapReduce.

Cluster Computing

In the domain of *cluster computing*, *Pramanick et al.* proposed System Recovery Benchmarking (SRB) framework for benchmarking availability of cluster computing systems [Mauro et al., 2004]. They measure dependability by injecting hardware faults such as node crashes on some nodes of the cluster. Finally, they measure the recovery time of a cluster in the event of a cluster node failure to estimate the dependability levels. Some of the interesting features of SRB are repeatability; and portability across data center clusters.

Cloud Computing Systems

More recently, we have found works that target benchmarking for *cloud computing systems*. YCSB [Cooper et al., 2010] is a performance benchmarking solution for evaluating different data storage systems in cloud environments. *Huppler et al.* and *Alexandru et al.* also proposed initial ideas to extend TPC benchmarks for cloud computing systems [Huppler, 2012, Alexandrov et al., 2012]. Though dependability is not the focused attribute in their works, yet they covered some elements of dependability such as availability and reliability. Moreover, these works presented the design challenges for developing benchmarks for cloud computing systems.

In Table 2.1, we provide a summary of state of art dependability benchmarks in various application domains of computer systems.

2.3 Existing Approaches of Performance Benchmarking

Benchmarks are needed to compare the performance and other quality aspects of different cloud services. Performance evaluation is a key step and an important criteria to adopt, choose or migrate a cloud service. In this section, we discuss the existing work on performance benchmarking in the domain of cloud computing services and also in other domains of computing such as web

Table 2.1: Classification of dependability benchmarks for various application domains

Domain	Reference	Dependability Metrics	Faultload	Workload / Dataload
Hardware	[Brown and Patterson, 2000]	availability (service unavailability), performance (response time, throughput)	single-fault workloads (disk sector write error), multi-fault workloads (disk failure in RAID system followed by replacement of failed disk followed by write failure while reconstructing array)	SPECWeb99
	[Friginal et al., 2011]	performance and dependability (execution time, availability and safety)	hardware	UDP Constant Bit Rate (CBR) data flows of 200 Kbps
Operating Systems	[Koopman et al., 1997]	robustness (# OS crashes, # hanging tasks, # abnormally terminated tasks)	software (operating system calls; rather than injecting faults)	read(), write(), open(), close(), fstat(), stat(), and select() system calls with a set of parameter values such as FILE HANDLE, BUFFER, LENGTH. example: read() is tested with all combinations of: 7 different file handle test cases, 9 different memory buffers, and 8 different lengths, for a total of 7x9x8 = 504 test cases
	[Barbosa et al., 2011]	dependability (arithmetic mean of register errors and memory errors)	hardware (single bit-flips in memory), software	six workloads: CRC32, Altimeter, Hamming Distance, Bit Count, LU Decomposition, Merge Sort
	[Kanoun et al., 2005]	robustness (# exceptions, # error codes, reaction time, restart time)	software (corrupted parameters of system calls (make use of existing tools))	PostMark (a file system performance benchmark)
Database Systems	[Vieira and Madeira, 2003]	performance (throughput), availability (time for which system is able to respond to at least one terminal within the minimum response time for each transaction) , data integrity (# data errors)	operator, software and hardware faults (abrupt OS shutdown, abrupt transactional engine shutdown, kill set of user sessions, delete table, delete user schema, delete file from disk, delete set of files from disk, delete all files from one disk)	TPC-C workload
Web Servers	[Durães et al., 2004]	dependability (autonomy, availability, accuracy, throughput, response time)	software, hardware and network	SPECWeb99 (performance benchmark representing typical requests submitted to real web-servers)
Web Services	[Vieira et al., 2007]	robustness (# application server corrupts/machine crashes or reboots, # web-service execution hangs, # abnormal terminations, # incorrect error code returns/delayed responses)	software (invalid call parameters: all SOAP messages sent by the emulated clients (generated by the workload emulator component) to the server are intercepted by the fault injector)	two different implementations of web services specified by the standard TPC-App performance benchmark Set of valid web-service calls
Cluster Computing	[Mauro et al., 2004]	availability (recovery time)	hardware (node crash)	SPECsfs and Postmark (set of standard filesystem benchmarks)
Cloud Computing	[Cooper et al., 2010]	availability, reliability	software	update heavy workload, read heavy workload

services, database systems, parallel computing and MapReduce. Following, we present a summary of state of art approaches of performance benchmarking and their limitations in the context of cloud computing services.

Virtualized Systems

TPC formed a subcommittee in 2010 to develop TPC-V, a publicly-available benchmark for virtualized databases. In [Bond et al., 2013], authors presented the work in progress *TPC-V*. In this paper, authors presented a reference architecture for this benchmark and some preliminary experiments for performance evaluation. VMmark [VMm, 2013] is another industry standard benchmark for server consolidations. It was developed by *VMware* for its vSphere hypervisor operating system; although it is possible to run VMmark on other hypervisors. The current VMmark version i.e. 2.0 adds platform-level workloads such as dynamic VM relocation (vMotion) and dynamic datastore relocation (storage vMotion) to traditional application-level workloads.

Database Systems

There are several major domain-specific database benchmarks from the Transaction Processing Performance Council (TPC) – in particular TPC-W for web e-commerce applications. There are a number of application benchmarks that measure the overall performance of a DBMS. The 001 Benchmark (commonly referred as the Sun Benchmark) was the first widely accepted benchmark that attempted to predict DBMS performance for engineering design applications [Cattell and Skeen, 1992, Duhl and Damon, 1988]. Because of its early visibility and its simplicity, it became a *de facto* standard for benchmarking of object oriented database systems. The primary motivation for the design of this benchmark is to focus upon engineering applications such as computer-aided design (CAD) and computer-aided software engineering (CASE). The benchmark is designed to be applied to object-oriented, relational, network or hierarchical database systems. The workload consisted of three operations: lookup, traversal, and insert. Response time was used as the performance metrics in their evaluation.

There is an increasing demand for adoption of XML database technology in commercial enterprises. Common industry sectors using XML database technology includes finance and banking, insurance, government, retail, health care and manufacturing. All major relational database systems offer some form of XML support. Therefore, most of the performance benchmarks are oriented towards XML based data management systems [DeWitt and Gray, 1992]. Following we discuss few of these benchmarks: XBench is a performance application benchmark [Yao et al., 2004]. XBench workload consisted a set of 20 SQL query types such as sort, join, retrieve etc. The performance metrics used in this benchmark is *Query Execution Time (in Milliseconds)*.

XOO7 was derived from OO7, which was designed to test the efficiency of object-oriented DBMS [Bressan et al., 2003, Carey et al., 1993]. The database model of XOO7 was mapped from that of OO7. Besides mapping the original 7 queries of OO7 benchmark into XML, XOO7 also added some XML specific queries. This is a publicly available benchmark which is implemented in C++ and supports multi-user operations. The performance metrics that this benchmark used is response time.

Nicola et al. [Nicola et al., 2007] developed an application-oriented and domain-specific benchmark called "Transaction Processing over XML" (TPoX). It exercised all aspects of XML databases, including storage, indexing, logging, transaction processing, and concurrency control. TPoX simulated a financial multi-user workload with XML data that consisted of transactions such as placing a new order, adding a customer, removing a customer and updating the status of an order. The system under test (SUT) included the database system, the operating system, the workload driver, and the hardware of the database server including storage and all auxiliary components. The primary performance metric of this benchmark was throughput, measured as TTPS (TPoX Transactions Per Second).

XMach benchmark consisted of a workload with eight queries and three update operations [Böhme and Rahm, 2001]. This benchmark was developed at the University of Leipzig. The goal of this benchmark was to test the number of queries that can be processed by the database per second, including the cost of processing. Additional measures included response time, bulk load times and database or index size. The main objective of this benchmark was to stress-test XML systems under a multi-user workload.

Schmidt et al. [Schmidt et al., 2001, Schmidt et al., 2002] provided a benchmark framework to assess the abilities of an XML database to cope with a broad range of different query types typically encountered in real-world scenarios. The benchmark was intended to help both developers and users to compare XML databases in a standardized application scenario. Their workload consisted of a set of 20 queries where each query is intended to challenge a particular aspect of the query processor. The performance metrics used in this benchmark was the *total running time of a query*.

Web Services

Web services technology enables a standards-based distributed computing platform built upon Web technologies including HTTP and XML. Several studies examine performance of Web services making use of benchmarking techniques. Few of them are discussed here.

The TPC-W is a transactional web benchmark designed to mimic operations of an e-commerce site. The TPC-W workload consisted of a set of web

interactions. The TPC-W workload explored a range of system components together with the execution environment. Like all other TPC benchmarks, the TPC-W benchmark specification was a written document which defines how to setup, execute, and document a TPC-W benchmark run [Menasce, 2002]. In [Chung and Hollingsworth, 2004], authors made use of TPC-W benchmark to choose an optimal configuration for a cluster based web service system. Throughput measured as Web Interactions per second (WIPS) and cost measured as Dollars per WIPS were the primary metrics in this benchmark.

Authors of [Head et al., 2005] presented a benchmark suite for quantifying, comparing, and contrasting the performance of Simple Object Access Protocol (SOAP) based web service implementations. SOAP is the most commonly used Web services communication protocol for information exchange in a distributed and heterogeneous environment. The primary metrics that they used to evaluate performance was latency. The architecture details were not provided and prototype was not publicly-available. Wickramage et al. [Wickramage and Weerawarana, 2005], introduced a benchmark that simulates the real world business services and a performance model to analyze Web service frameworks. The benchmark consisted of various real world scenarios such as credit card processing, online store operations, transactions between travel agents etc. This benchmark is implemented as an open source project.

Zhu et al. [Zhu et al., 2006] presented a model-driven approach to generate a benchmark application for web service platforms. In this benchmark, users had to implement their own workloads and generate the stress data manually. They used the standard performance metrics such as web service interaction response time (the time taken to perform a successful web interaction) and service interactions per second. Mizouni et al. [Mizouni et al., 2011] first proposed an architecture that allows the deployment of Web Services on mobile devices. Then, they evaluated the QoS of these web services. In this study, authors considered the following QoS parameters: throughput, availability, response time, and scalability. Rosenberg et al. focused on the QoS evaluation of web services [Rosenberg et al., 2006]. Their metrics consisted of latency; response time; execution time; availability; and accuracy. Stantchev [Stantchev, 2009] made use of WSTest benchmark to evaluate windows instance in Amazon EC2. A virtual client generates requests consisting of a SOAP requests were sent to one of the instances running WSTest in Amazon EC2. The performance metric that they used is throughput measured as transaction rate.

Parallel Computing

In the domain of parallel computing, several benchmark suite implementations are available for general purpose multi-core architectures. Standard

Performance Evaluation Corporation (SPEC) [SPE, 2013] and Embedded Microprocessor Benchmark Consortium (EEMBC) [EEM, 2013] are two corporations that developed widely used benchmark suites for evaluating general purpose CPUs and embedded processors, respectively. SPEC CPU2006 focused upon compute-intensive workloads for scientific and engineering applications. SPLASH-2 [Woo et al., 1995] consisted of multi-threaded scientific applications and applications belonging to computer graphics.

Rodinia is a benchmark suite for parallel computing that supports GPU platforms [Che et al., 2009]. The workload exhibited various types of parallelism, data-access patterns, and data-sharing characteristics. It covered applications from diverse application domains such as data mining, grid computing and MapReduce. Other parallel benchmark suites included MediaBench [Med, 2006] and ALPBench [Li et al., 2005] for multimedia applications, and BioParallel [Jaleel et al., 2006] for biomedical applications.

In the related domain of grid computing, an interesting benchmark is NAS Grid Benchmark (NGB) [Frumkin and Van der Wijngaart,]. NGB are representative of tasks typically executed on the Grid, and also specify well-defined, measurable quantities of workload. To verify the correctness of the benchmark results, NGB provided reliable verification testing. NGB also measured to a small extent, the data transfer capabilities of the communication network particularly latency and bandwidth. NGB do not measured security and fault-tolerance capabilities; despite the fact that they are the primary attributes of a grid computing system.

MapReduce Systems

Works have more specifically studied MapReduce performance benchmarking, such as HiBench [Huang et al., 2010], MRBench [Kim et al., 2008], PigMix [Pig, 2011], Hive Performance Benchmarks [Pavlo et al., 2009]. GridMix3 [Gri, 2011], and SWIM [Chen et al., 2011] are useful tools for Hadoop MapReduce performance benchmarking. However, these works do not incorporate multi-user workloads and therefore do not exhibit real-world scenarios. HiBench consisted of eight MapReduce jobs [Huang et al., 2010]. The benchmark measured performance in terms of job processing time, MapReduce task throughput, and I/O throughput. While HiBench included different types of jobs, it did not support concurrent job execution, that is, the whole MapReduce cluster is dedicated to a single job at a time, which inhibits cluster consolidation. Thus, it failed to capture different workloads and job arrival rates. Furthermore, HiBench did not consider faultload injection and did not allow the evaluation of MapReduce dependability.

MRBench is a domain-specific benchmark that evaluated business-oriented queries [Kim et al., 2008]. It used large datasets and complex MapReduce

queries derived from TPC-H [TPC, 2011a]. However, as HiBench, MRBench failed to capture job concurrency and arrival rates, workload variations, and it did not evaluate MapReduce dependability in presence of failures.

Similarly, PigMix and Hive Performance Benchmark used a set of queries to specifically track the performance improvement of respectively Pig and Hive platforms [White, 2009]. Pig and Hive run on top of Hadoop MapReduce, the former provides a high-level language for expressing large data analysis, and the latter is a data warehouse system for ad-hoc querying.

GridMix3 took as input a job trace from a specific workload and emulated synthetic jobs mined from that trace [Gri, 2011]. GridMix3 is able to replay synthetic jobs that generate a comparable job arrival rate and a comparable load on the I/O subsystems as the original jobs in the specific workload did. However, GridMix3 does not capture the processing model and the failure model from the traces. Thus, it fails to reproduce comparable job processing times and failures in the MapReduce cluster.

SWIM is a similar framework that synthesizes specific MapReduce workloads [Chen et al., 2011]. The framework first samples MapReduce cluster traces, and then executes the synthetic workloads using an existing MapReduce infrastructure to evaluate performance. Here again, the proposed framework does not capture job failures and does not model the dependability of the MapReduce cluster.

Cloud Computing Systems

Researchers used benchmarks in order to compare the performance of a cloud service such as IaaS. In their position paper, Iosup et al., focued on the IaaS cloud specific elements of benchmarking, from a user's perspective [Iosup, 2013]. They also discussed various research challenges in designing a generic approach for IaaS cloud benchmarking. High Performance Computing Challenge (HPCC) benchmarks are used to evaluate performance aspects of cloud services [HPC, 2011]. HPCC benchmark suite further consisted of 7 benchmarks. *Mehrotra et al.* made use of HPCC benchmark to evaluate the performance of Amazon EC2 service [Mehrotra et al., 2012]. In this study, authors focused upon the computational performance and network communication between the cluster nodes.

Ostermann et al. [Ostermann et al., 2010] evaluated the performance of different types of virtual machines (VMs) provided by EC2 using HPCC benchmark among others. There are various other works of use of benchmarks to evaluate the quality aspects of cloud services in various application domains. Authors compared and evaluated the features and performance of open-source solutions in supporting Geosciences [Huang et al., 2013]. This study included the comparison of three open-source cloud solutions, including

OpenNebula, *Eucalyptus*, and *CloudStack* for performance aspects of Geoscientific applications.

In [Lee et al., 2009] authors argued that conventional frameworks do not effectively support the evaluation of SaaS-specific quality aspects. They proposed a quality model for evaluating SaaS applications. They defined ten metrics for evaluating the quality attributes such as *reliability; efficiency; scalability and reusability*. Such models could be used by cloud service providers to evaluate their services and also predict their *Return on Investment (ROI)*. One more similar work is by Tsai et al. [Tsai et al., 2011], where authors focus on scalability aspects of cloud services. In this work, they discussed a quality model for scalability evaluation and provide relevant metrics to measure this. Kossman et al. [Kossmann et al., 2010] used TPC-W (a transactional web e-Commerce benchmark) and provided a comprehensive evaluation of existing commercial cloud service providers for transaction processing.

The Yahoo Cloud Serving Benchmark (YCSB) was designed to deliver a database benchmark framework for evaluating the performance of cloud serving systems. The YCSB framework and workloads are available as open source to evaluate systems. YCSB has been used to evaluate performance and elasticity of four systems i.e. *Cassandra*, *HBase*, *PNUTS*, and a simple shared MySQL implementation [Cooper et al., 2010]. This framework is also intended to evaluate other aspects such as availability and replication by supporting easy creation of workload. Authors also discussed the trade-offs between various aspects of performance such as read vs. write performance. However, we observed that, authors did not include any faultloads in their benchmark. Without the inclusion of a representative faultload, evaluation of availability and other dependability aspects seems to be unrealistic.

Cloud storage services are becoming increasingly accepted as replacements for traditional file systems. It is important to effectively measure the performance of these services so that users of these services can easily compare and evaluate different services. Authors at *Intel* presented a benchmark named COSBench (Cloud Object Storage Benchmark) [Zheng et al., 2012]. This benchmark tool is still under development and is intended to be used for cloud object storage services. Object storage services provide interfaces to store and access files in a way that is similar regular file systems. In addition, these services are often characterized by what is lacking in traditional technologies: scalability, cost-effectiveness, and ease-of-use. These services either rely on public services, such as Amazon Simple Storage Service (Amazon S3) or manage to build their own private clouds with the help of open source solutions. COSBench evaluates the performance aspects of these services using realistic workload patterns. In [Agarwal and Prasad, 2012], authors presented

an open-source benchmark suite named as *AzureBench*. This benchmark is used in the performance analysis of Microsoft Azure cloud platform's storage services. In Table 2.2, we provide a summary of state of art performance benchmarks in various application domains of computer systems.

2.4 Discussion

In this chapter, we studied the existing work on dependability and performance benchmarking not only in the domain of cloud computing services but also in other domains of computing such as web services, database systems, hardware and cluster computing. Our analysis of state of the art approaches bring out the fact that there are very few standard benchmarks designed for cloud computing services. However, there exists standard benchmarks in other domains of computing.

To date, there has been no benchmarking approach to measure the dependability and performance of large-scale cloud services. For some specific examples of such services, few benchmarks emerged to measure various aspects of performance. However, there is a severe mismatch between the diversity of real life use cases, and a narrow coverage of behaviours by such benchmarks. Of course, the problem is less severe for established application domains that have evolved to become large-scale cloud services.

Fortunately, prior studies have also pioneered in developing a set of such techniques for different application domains [DBe, 2004]. These techniques provide us a basic to design dependability and performance benchmark for complex and distributed cloud service. These techniques rely on complex workloads and dataloads for a range of application domains such as database systems, web services and operating systems. The rest of this dissertation applies these techniques to large-scale cloud services to design and develop dependability and performance benchmarks.

Analysis of the state of the art benchmarking approaches motivates us to come up with a comprehensive benchmark suite for evaluating the dependability of cloud services that achieves following important objectives:

- **Multi-criteria analysis.** Benchmark should measure and analyze the performance *and* dependability of cloud services. In particular, it should consider several measurement metrics such as reliability, availability, financial cost, request response time (i.e. latency), and request throughput.

- **Diversity.** Benchmark should cover a variety of application domains and programs with a wide range of computation characteristics. It

Table 2.2: Classification of performance benchmarks for various application domains

Domain	Reference	Performance Metrics	Faultload	Workload / Dataload
Virtualized Systems	[Bond et al., 2013]	throughput	no faultload	CPU heavy and I/O heavy transactions upon virtual machines
Database Systems	[Cattell and Skeen, 1992]	response time	no faultload	seven database lookup operations: Name Lookup, Range Lookup, Group Lookup, Reference Lookup, Record Insert, Sequential Scan, Database Open
	[Nicola et al., 2007]	throughput (Transactions per second)	no faultload	multi-user read/write workload on database, with 70 % read and 30 % write operations
Web Services	[Menasce, 2002]	throughput (Web interactions per second)	no faultload	E-commerce workload such as search, add etc.
	[Head et al., 2005]	Latency	no faultload	operations in WSDL files
	[Wickramage and Weerawarana, 2005]	response time	no faultload	real world business services such as credit card processing, online store operations etc.
Parallel Computing	[Che et al., 2009]	power consumption	no faultload	workloads exhibiting parallelism, data-access patterns, and data-sharing characteristics such as applications based upon data mining, image processing
MapReduce Systems	[Gri, 2011]	response time	no faultload	emulates synthetic jobs using job traces from a specific workload
	[Huang et al., 2010]	response time, IO utilization, network throughput	no faultload	sort, search, machine learning etc.
	[Kim et al., 2008]	response time, throughput	no faultload	complex MapReduce queries derived from TPC- H
Cloud Computing Systems	[HPC, 2011]	throughput, latency and communication bandwidth	no faultload	scientific computing applications
	[Cooper et al., 2010]	latency, throughput	hardware faults, software faults, network faults	read-heavy workloads, write-heavy workloads, scan workloads, etc.
	[Zheng et al., 2012]	response time, throughput and network bandwidth	no faultload	different workload models for cloud storage service

should consider batch applications as well as interactive applications. Moreover, it should allow to characterize different aspects of application load such as the *faultload*, the *workload* and the *dataload*.

- **Usability.** Benchmark must be easy to use, configure and deploy on a distributed cluster.

Benchmarks must be designed for diversity and multi criteria analysis. This challenge arises from the simple fact that the cloud services being studied are complex, and therefore workloads, dataloads and faultloads need to be described in multiple ways, i.e., multiple features, characteristics, or dimensions. For example, a large-scale data processing MapReduce service could be used for different application domains such as data mining, machine learning, bioinformatics among others. In reality, the workload may consist of many complex minority behavior patterns. For example, some application could be heavy based on their computational characteristics while some could be heavy based on the data access characteristics. Also there could be few, which could be heavy both according to the computational and data access characteristics.

Therefore, diversity is a must have property of a benchmark that is designed to evaluate the dependability and performance of a cloud service. Multi criteria analysis is another property that is desired for benchmarking a complex system. This means that multiple attributes must be analysed together. As we discussed in this chapter, earlier works have mainly focused only on the performance aspects of cloud services while the dependability aspects are ignored.

To illustrate the inherent challenges of new computing paradigms and associated ways of benchmarking, we consider an example MapReduce service. The dependability and performance benchmark must be easy to use by the naive users of service. It must provide an easy way to inject workload, dataload and faultload in a running MapReduce cluster. The workload, dataload and faultload must represent real world services. Moreover, it should provide an easier way to analyze the system's dependability and performance. It should provide high level performance statistics such as response time and throughput for various applications. Similarly, it should provide high level dependability statistics such as number of failed jobs and number of successful jobs. There must be a mechanism to provide low level statistics such as number of failed MapReduce tasks. It should provide information about the data read and write throughput separately.

Existing benchmarks are quite helpful when considering the functionality and scalability concerns in specific application domains. However, they are limited in design and functionality to address the concern of large scale cloud

services such as MapReduce. Consequently, it becomes a pressing concern to design benchmarking approaches to better understand the dependability and performance of cloud computing services.

Chapter 3

Towards a General Architecture for Dependability & Performance Benchmarking

Contents

3.1 Introduction **37**

3.2 Objectives & Approach **37**

3.3 General Definitions & System Model **41**

3.3.1 Workload 42

3.3.2 Faultload 42

3.3.3 Dataload 43

3.3.4 Dependability & Performance Analysis 43

3.4 Design Principles **45**

3.4.1 Architecture Overview 45

3.4.2 General Software Framework 46

3.5 On the Generality of the Proposed Architecture . . . **49**

3.5.1 Reduced development costs 51

3.5.2 Better usability 52

3.5.3 Higher adaptability 52

3.6 Summary **52**

3.1 Introduction

With the increasing demand and benefits of cloud computing services, new solutions are needed to benchmark the dependability and performance of these services. Designing a dependability and performance benchmark that covers a variety of fault and execution scenarios, poses various architectural challenges. Most of the state of art approaches of dependability and performance benchmarking do not address the specific challenges for cloud services. Hence, there is a strong need to develop dependability and performance benchmarking solutions for cloud services.

In this chapter, we present a generic software architecture for dependability and performance benchmarking for cloud computing services. We provide the details of this generic architecture i.e. various components and modules, that are responsible for injecting faults in cloud services in addition to the components responsible for measuring the performance and dependability. We make use of this architecture to build two software prototypes: MRBS and MemCB. These prototypes are used to benchmark two popular cloud services: MapReduce and Memcached. The case studies with the use of software prototypes developed as part of this thesis demonstrates the benefits of building a generic architecture. Having available a generic software architecture would help the designers of dependability benchmarking solutions in reducing the efforts to design and develop new benchmarks for such services.

3.2 Objectives & Approach

Guaranteeing reliability, availability and performance are one of the major challenges for cloud service providers such as Amazon, Google, Salesforce and Microsoft. Primary motivation for introducing a benchmarking framework for cloud services is that there is no scientific approach or framework so far in the existing literature that could help users to evaluate the important quality aspects such as dependability for cloud computing services. MapReduce is a well known example of such services that provides a convenient means for distributed data processing and automatic parallel execution in cloud computing environments.

Various benchmarking programs are often used to evaluate the specific frameworks such as Hadoop MapReduce [Dean and Ghemawat, 2004, Gri, 2011, Huang et al., 2010]. However, they are mainly micro level benchmarks measuring specific Hadoop properties. Most of the times, the cloud service users are more interested in knowing the performance and associated costs rather than measuring low level statistics such as CPU behaviour and memory. Here,

it is to be noted that we do not mean that low level system statistics are not important to analyze the quality aspects. Rather, we argue that as a cloud service are used even by naive users and for these users or service providers, most important question is how an application performs in a realistic environment (e.g. in the presence of failures) and what is the associated cost? Most of the time, high level statistics are necessary to get a simple idea of system's dependability and performance.

Dependability benchmarking is a promising approach to understand a system behaviour in the presence of faults. For a benchmark suite to enable a thorough analysis of a wide range features of cloud service, it must provide the following. First, it must enable the empirical evaluation of the performance and reliability of cloud services, two key properties of a cloud service. Furthermore, with the advent of pay-as-you-go model, a benchmark suite must allow the evaluation of the costs of the running a representative workload in cloud environments. Second, it must cover a variety of application domains and workload characteristics, ranging from compute intensive to data-intensive applications, online real-time applications as well as batch-oriented applications. Moreover, in order to stress reliability and performance, the benchmark suite must enable different fault injection rates, and it must allow the generation of different workloads (i.e. #clients, clients request distribution).

- a variety of application domains
- a variety of application types such as real-time applications vs. batch-oriented applications [Neumeyer et al., 2010], [App, 2011]
- a wide range of workload characteristics such as application load (i.e. how many users concurrently access the application, mono-user systems vs. multi-user systems), data-intensive vs. compute-intensive workloads, failure frequency of cluster nodes
- the benchmark suite must not depend on any particular platform (e.g. public or private cloud)

Building complex benchmarks such as dependability benchmarks is difficult, since these benchmarks are composed of diverse fault types, different ways to inject the faults in a running system and diverse metrics to understand the dependability levels. New services such as cloud computing services add an additional layer of complexity because of virtualization and sharing of the resources.

DBench report introduced following key features for a dependability benchmark [DBe, 2004].

Representativeness The measures, the workload and the faultload should be as representative as possible. The workload representativeness reflects how well the workload of the benchmark corresponds to the actual workload of the real system. Fault representativeness is a measure of how well injected faults correspond to real faults, i.e., faults affecting systems in real use.

Repeatability and Reproducibility Repeatability is the property, which guarantees statistically equivalent results when the benchmark is run more than once in the same environment. Reproducibility is the property, which guarantees that another party obtains statistically equivalent results when the benchmark is implemented from the same specifications and is used to benchmark the same SUB

Portability Portability refers to the applicability of a benchmark specification to various target systems within a particular application area. A portable benchmark can be implemented and run on various target systems within the application area.

Non-intrusiveness If the implementation of the benchmark introduces changes on the system under test (either at the structure level or at the behavior level) it means that the benchmark is intrusive. The benchmark must require minimum changes in the system under test.

Scalability A benchmark must be able to evaluate systems of different sizes.

Benchmarking Time and Cost The benchmarking time is the time required to obtain the result from the benchmark. It consists of three parts: i) set-up and preparations, ii) running the actual benchmark program (i.e., execution time) and iii) data analysis. A key goal of dependability benchmarking is to provide an efficient and cost effective approach to characterize dependability.

Similarly, *Gray et al.* [Gray, 1992] identified four criteria for a successful benchmark:

- Relevance to an application domain.
- Portability to allow benchmarking of different systems.
- Scalability to support benchmarking large systems.
- Simplicity so that results are understandable.

As discussed in Chapter 2, dependability benchmarks have been proposed in various domains of computing such as hardware; cluster computing; operating systems; database systems; web servers; and web services among

others [Mauro et al., 2004, Brown and Patterson, 2000, Barbosa et al., 2011, Vieira and Madeira, 2003, Durães et al., 2004]. Various works aim to propose a standard approach for dependability and performance benchmarking. **DBench** project aims to provide a basic foundation for dependability benchmarking and performs a detailed study on dependability benchmarks [DBe, 2004]. *Barbosa et al.* [Barbosa et al., 2011] introduced a dependability benchmark to evaluate dependability of operating systems. *Vieira et al.* [Vieira and Madeira, 2003] proposed a dependability benchmark for Online transaction processing (OLTP) systems. *Duraes et al.* [Durães et al., 2004] present a benchmark for the dependability of web-servers.

One of the limitations of these approaches is that they do not discuss a generic software architecture that can be extended by the users to develop dependability benchmarks for their systems. A generic architecture might be a key factor in reducing cost to build a new benchmark and also improve the overall quality of benchmarking process. This is possible using generic components which can be reused in building the new benchmark. In addition, it must be able to support a diverse set of workload and fault injection scenarios. Of course, the benefits of reuse would be maximum if the new benchmark to be built is closer to the application domain of the previous benchmark. For example, using the case studies conducted in this thesis, we demonstrate two software prototypes built using the generic architecture in PaaS domain of cloud computing.

We also observed that state of the art dependability benchmarks do not address the challenges raised by cloud computing systems. This is mainly due to the fact that cloud computing is a relatively new domain. Some of the terms such as faultload, workload, and dataload are not clearly defined for cloud computing use cases. The literature lacks the specific knowledge to build a benchmark for cloud computing services. Therefore, it is not possible to extend any existing benchmark easily for a cloud computing service. However, we argue that the general definitions and principals can still be extracted from the existing works and adapted to build a generic dependability and performance benchmarking architecture for cloud computing services.

While designing a generic architecture for cloud computing services, we define the following objectives:

1. Building a system model and identifying the general terms used in the benchmark such as for faultload, workload, and dataload.

2. Introducing a software architecture and defining its generic components and modules. In addition to this, inferring relationships between various components.

3. Demonstrating the benefits of architecture by building various software prototypes using the architecture and making maximum component and code reuse.

In this chapter, we make following contributions:

1. We discuss various components of a dependability and performance benchmark in detail. The major components of the dependability and performance benchmark architecture, i.e. workload, faultload, statistics analysis, etc. are generic and can be easily adapted to build a new dependability and performance benchmark.

2. We provide the details of various classes of our architecture API. The classes can be easily adapted for designing a benchmark for a new service.

3. We provide a detailed illustration of instantiating the proposed architecture in the domain of cloud service. We choose the two widely used cloud service models: MapReduce and Memcached service. The proposed architecture is used in each of these service models to define the components of the dependability and performance benchmark such as faultload, injection of the faultload and measuring the performance and dependability.

The proposed architecture can be used by the designers of dependability and performance benchmark solutions for cloud computing services. The software prototypes MRBS and MemCB can be directly used to analyse dependability and performance of MapReduce systems and Memcached service respectively. This chapter is further organized as follows. Section 3.3 discusses the system model and general definitions. Section 3.4 describes the proposed generic architecture and software framework of dependability and performance benchmarking. In Section 3.5, we instantiate the proposed architecture for cloud services. In Section 3.6, we present a summary of this chapter.

3.3 General Definitions & System Model

Figure 3.1 depicts the various components of dependability benchmark architecture.

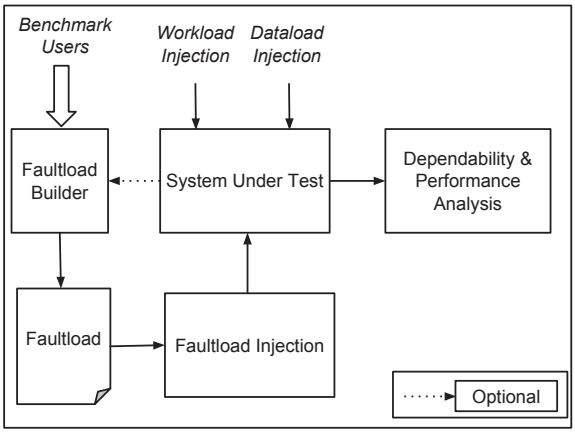


Figure 3.1: High level architecture of dependability and performance benchmark

3.3.1 Workload

The primary goal of a benchmark is to emulate a particular kind of real world workloads widely used in industrial applications on a system. There are two categories of a workload: synthetic workload and real workload. In the case of a synthetic workload, a program is responsible for imposing the workload on the system. However, a real workload gives a better measure of performance and dependability by running real applications in the benchmark. For example, the workload in MRBS are real applications. Moreover, all the workloads in MRBS are also publicly available.

The workload is characterized by the benchmark application to execute, and the number of concurrent clients issuing requests on that benchmark application. The workload is also characterized by the execution mode which may be interactive or batch. In interactive mode, concurrent clients share cluster resources (i.e. have their requests executed) at the same time. The workload is also characterized by client request distribution, that is the relative frequencies of different request types. It may follow different distribution laws (known as workload mixes), such as a random distribution.

3.3.2 Faultload

A faultload describes *what* fault occurs (e.g. a node crash), *where* it occurs (e.g. in which node of the cluster), and *when* it occurs (e.g. one hour after the application started). Such as faultload can be described in a file. An example of such a faultload file is provided in Figure 3.2.

The faultload consists of a set of faults. These faults can either be **hard-**

ware faults such as a node crash or **software faults** such as a run time error in a program. Sometimes a significantly stressful workload can also lead to failures. Therefore, this kind of workload can also be considered as a faultload. However, the case studies in this thesis incorporates primarily the hardware faults and software faults.

Our faultload satisfies the two important properties: representativeness and ease of use. The most desirable feature for a faultload is its representativeness since faults are intended to emulate the real threats, the system would experience. Secondly, the faults should also be easy to inject i.e. fault can be injected even by naive users of service.

There are different possible ways to build a faultload: A faultload description may also be automatically obtained, either randomly or based on traces of previous application runs; Users can explicitly build synthetic faultloads representing various fault scenarios; faultload can be random values. The fault injector component in Figure 3.1 is responsible for injecting the faultload to the system under test.

```
<TimeStamp, Fault-Type, Fault-Number >
<0, NodeFault, 2>
<90, SoftwareFault, 3>
...
```

Figure 3.2: An example faultload file

3.3.3 Dataload

Designing a benchmark for a cloud computing service demands that it supports processing of large data sets. There is a great demand for cloud computing services for computation in scientific applications. These applications normally have very large data to process [Iosup et al., 2011]. Therefore, the dataload of a benchmark for a cloud service must represent such a large dataset.

The real world applications might possess different behaviour according to its data and computational requirements. Therefore, the dataload is characterized by the size and also *nature* of data sets used as inputs for a benchmark application. Ideally, datasets used in dependability and performance benchmark should be real and publicly available.

3.3.4 Dependability & Performance Analysis

The analysis of the desired attributes such as **dependability** and **performance** is done using the information given by the benchmark. This informa-

tion is processed to compute the statistics. The statistics can be generated using files such as HTML files and/or charts. The benchmark contains the scripts to plot these charts. There might be different runs of a benchmark to compare the results. These are called scenarios. Some of the scenarios are emulated without injecting any fault to compare a faulty system with a base line system (where, no faults are injected). In addition to this, various metrics that demonstrate dependability and performance of a system must be defined. Some examples of dependability and performance metrics used in this thesis are:

Dependability Analysis

Dependability is composed of metrics such as reliability, and availability.

- **Reliability** is the ability of a service to support successful requests during a period of time. Reliability is the overall measure of dependability and it is related to the number of failures per unit of time interval. Reliability is measured as the ratio of successful client requests to the total number of requests, during a period of time.

- **Availability** is the measure of time a system is able to handle client requests successfully. Ideally, this should be 100%. This means that a system should be always available to handle successful business requests. However, in real world, the application might have downtimes because of number of clients reaching the maximum limit, network constraints, power outrages, server crashes etc.

  We measure availability from the client's perspective as the ratio of, on the one hand, the time the benchmark service is capable of returning successful responses to the client, and on the other hand, the total time. As for reliability, availability is measured during a period of time.

Performance Analysis

Performance and cost statistics can be generated using client request response time, request throughput, and the financial cost of a client request.

- **Throughput** is a measure of the amount of work an application performs in per unit of time. For business applications, work is typically measured in transactions per second (tps). We define throughput as the number of clients requests handled by the system per unit of time.

- **Response time** is a measure of the latency an application exhibits in processing a business transaction. Response time is calculated as the time an application takes to respond to some input. Ideally, the

response time should be shorter for business transactions. Moreover, a shorter response time for an application which is hosted in a cloud premise definitely leads to economic benefits for the end users. We measure response time as the time elapsed from the moment the client submits a request until the response is received completely by the client.

3.4 Design Principles

3.4.1 Architecture Overview

Figure 3.3 depicts the four main phases of a typical run of the proposed dependability and performance benchmark. These are: load generation phase, benchmarking phase, monitoring phase and statistics measurement phase.

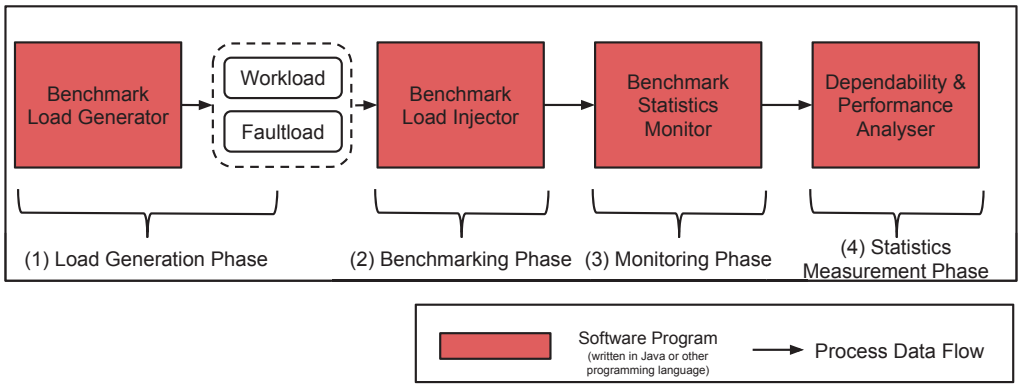


Figure 3.3: Phases of the benchmarking approach of the architecture

In the first phase i.e. load generation phase, workload and faultload specified by dependability benchmark user are generated. This phase might also include an optional dataload generation, if it is needed by any of the benchmark workload application. In Section 3.3, we define workload, faultload and dataload. Users of the benchmark must describe the loads that they want to inject during the benchmark run. Ideally this is described with configuration parameters and users have the options to choose between a range of these parameters. For example, a user might choose a dataload size of 1 GB to 100 GB. Similarly, for a faultload he/she can choose the faults that he/she wants to inject in the running system.

Once the loads are defined by the users, in the second phase i.e. load injection phase, they are injected to the system under test according to the time described by the users. Normally, there is a warm-up period before a run time period. Normally, the warm period is used for starting various

processes such as warming up caches, etc., so that the system is in a relatively steady state. This provides a better estimate of system's dependability and performance compared to a scenario where no warm-up period is used. After the warm-up period, the benchmark runs for a particular time as specified by the user. During this run time period, workload is run with the given dataload and faults are injected.

In the third phase i.e. monitoring phase, various statistics such as response time, throughput, availability, reliability etc. are computed and stored. During this phase, the system counters can be used to generate the desired statistics. For example, the number of failed jobs, number of successful jobs can be calculated to plot the reliability statistics. Some other example of statistics are provided in Section 3.3.4.

In the final phase i.e. statistics measurement phase, statistics outputs are produced in the form of user friendly graphs or files. During this phase, the statistics values obtained in the preceding phase are computed to plot easy to read charts/html files. These charts/html files provide an easy to understand view of system's performance and dependability.

3.4.2 General Software Framework

Figure 3.4 details the classes and methods that are developed to detail the high level architecture proposed in Figure 3.1. One of the most important benefit of this architecture is that we can leverage large scale reuse of the components. The design of most of the components of architecture is kept generic. As this can be observed from Figure 3.4, *faultload*, *faultload injector*, *benchmark*, *workload injector* are independent from the system under test. To build a new benchmark, these components do not need any modifications. The system dependent components such as *faultype*, *workload* and *SystemUnderTestAPI* are the ones where due to various dependencies it is difficult to reuse them. The architecture consists of generic classes such as faultload, benchmark, workload injector and statistics. The other classes can also be adapted easily according to the system under test. Various classes and associated methods of the generic architecture are explained as follows:

Faultload. This class includes method that reads the particular faultload and provide description of the locations of these faults. Faultload can be described using a file according to various criterion like *random, trace based* or *synthetic*. This class includes methods to generate the faultload based on the criteria specified by the user of the benchmark. For example, in the simplest case, if a user wants to inject a random faultload, this class invokes the method which is responsible for generating a set of random fault types,

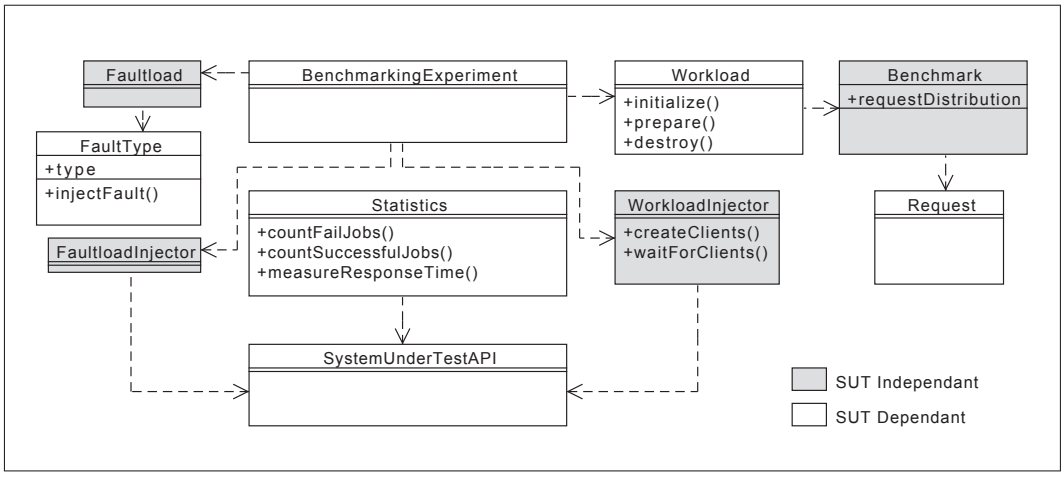


Figure 3.4: Various classes of dependability and performance benchmark architecture

timestamps etc. in the faultload file (as shown in Figure 3.2). This class has also few other methods. For example, method that specify target nodes in the cases where a node crash fault needs to be injected.

FaultloadInjector. This class includes a method that injects a given faultload in a running system. This class also consists of methods that monitors the time of running of the benchmark. As we mentioned earlier, a benchmark run might be optionally composed of a period of warm up time and run time. This class is responsible to inject the faults during the run time of the benchmark. For example, it will read the timestamps and fault types in the faultload file and would inject a particular fault type at the specified time.

Workload. This class includes methods to initialize, prepare and destroy a benchmark. As we discussed earlier, a dependability benchmark needs an application that a user run on the top of dependability benchmark. The workload class includes specific methods that are needed to start these applications, run for the given run time of the benchmark and stop when the time to run the benchmark is over. There might be some optional methods to prepare any input data, if it is needed by the benchmark workload application.

WorkloadInjector. According to the definition of the workload in Section 3.3.1, this class contains methods to send client requests that are described by the workload. In case, where a user wants to emulate a multi-clients behaviour, this class would create the concurrent clients that will send requests. In a simplest First in First out (FIFO) case, there will be a method to create concurrent clients, wait for the completion of the request (or the end of run time of benchmark if it comes before) and send the next request.

Benchmark. A workload might consist of a number of benchmark appli-

cations. The client can send requests according to different rules and combinations. For example, in FIFO case, there will be method to create concurrent client, wait for the completion of the request (or the end of run time of benchmark if it comes before) and send the next request. The next client request may be chosen either randomly or according to a probability distribution.

This is explained as follows: A benchmark consists of a set of client requests. This class contains request distribution functions. The requests might be uniformly distributed or some requests may have higher probabilities of occurrence than others. This might be specified in a transition matrix for all the requests in the benchmark. The motivation for this is that all these client requests can be different in terms of data and computation behaviour. Therefore, to get a better understanding of system's dependability and performance, a user can set the probability for the occurrence of the next client request.

Request. A benchmark may consists of one or more applications. Benchmark class calls the *Request* class for issuing a new client request.

Statistics. This class contains methods for calculating the statistics. For example, it includes methods that count the number of successful and failed jobs to measure the availability and reliability that are used to build the dependability and performance metrics. Moreover, it is also responsible to invoke methods that measure the response time of different client requests. As we mentioned in Section 3.3.4, response time and throughput are used to build the performance metrics. To get a better understanding, statistics for failed requests are analysed separately than the successful requests. In addition, this class also has methods to monitor the time for upload of the data. Although data upload time is not used to generate the dependability and performance metric.

BenchmarkingExperiment. This is the main class which is primarily responsible for uploading the data, and calling other methods of classes discussed before. This class orchestrates all the classes to perform the operation as specified by the user in the configuration file.

SystemUnderTestAPI. This class is responsible of injecting the workload and faultload into the system under test. This class contains methods that identify the nodes in the distributed system. It might also differentiate between master and slave nodes, e.g. in the case of Hadoop cluster or server and clients in the case of Memcached system. The identification of nodes is important because a user might want to select where he/she wants to inject a particular fault. For example, in the Hadoop version 1.0.0, master node is not fault tolerant [Had, 2013]. Therefore, he/she might want to skip the master node for fault injection.

Moreover, this class has methods that are responsible for starting and stopping the nodes at the beginning and at the end of a benchmark respectively. There might be some additional services that might need a start and stop such as Hadoop services. This class also communicates with *Statistics* class to send the information from system counters to build the required statistics.

3.5 On the Generality of the Proposed Architecture

Consider a scenario where a software designer wants to develop dependability and performance benchmark for a cloud service. This benchmark must provide the following functional and non-functional properties:

- It should provide a means to create a faultload.
- It should provide a means to inject faults in a running system.
- It should provide a means to create a workload and dataload.
- It should provide a means to measure the dependability and performance levels using metrics such as reliability, availability, and throughput.
- The benchmark must provide features such as Portability, Representativeness, Repeatability and Reproducibility among others.

In this thesis, we conduct two case studies to validate the general architecture proposed in this chapter. Using the general architecture introduced in this chapter, we develop dependability and performance benchmarks for two widely used big data cloud services, MapReduce and Memcached. MapReduce is a relatively young programming model and run-time system for large-scale data processing [Dean and Ghemawat, 2004]. It provides a convenient means for distributed data processing and automatic parallel execution on clusters of machines. We developed MRBS [Sangroya et al., 2012a], which is a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems. MRBS includes five benchmarks covering several application domains and a wide range of execution scenarios such as data-intensive vs. compute-intensive applications, or batch applications vs. online interactive applications. MRBS can be used to evaluate the performance and/or the dependability of MapReduce systems, using various workloads, dataloads, and faultloads. MRBS benchmark suite consists of benchmarks from five application domains: recommendation systems, business intelligence, bioinformatics,

text processing, and data mining. MRBS supports injection of various faults belonging to one of the fault types handled by Hadoop MapReduce. Hadoop is able to tolerate failures of different types such as node crash, task process crash, task software fault and hanging tasks. In Chapter 4, we present more details of using MRBS to evaluate the performance and dependability of a MapReduce service.

Furthermore, we develop MemCB for evaluating the dependability and performance of Memcached systems. Memcached is a high-performance, distributed caching system which is commonly used to speed up dynamic web applications by lightening the database load. This is done by speeding up the access to databases by storing the results of the previous database computations or any other data which is accessed very often. Memcached works like a giant hash table distributed across multiple machines. When the table is full, subsequent inserts cause older data to be purged in least recently used (LRU) order. Memcached is easily deployable over existing applications. Memcached is used by high-traffic websites such as Youtube, Wikipedia, Facebook and others [Mem, 2013b].

MemCB provides a simpler way for faultload injection in a running Memcached cluster. This covers different fault types, such as node crashes and network faults, injected at different rates, that provide a means to analyze the fault-tolerance under different scenarios. MemCB includes a workload generation toolkit that emulates concurrent client requests in a Memcached cluster. The faultload for Memcached consists of node crashes and network errors. MemCB workload consists of multiple concurrent clients issuing requests to the Memcached system. We use two kinds of client requests. The *set* requests that write and store our generated data in Memcached; the *get* requests that retrieve the data from Memcached. Dataload of MemCB is composed of keys. Each key in the dataload is a concatenation of a $prefix and $key_id. Users can decide to modify the values of key prefix and key ID. MemCB produces runtime statistics related to performance and dependability, such as Response Time, Throughput, and reliability. Chapter 6, presents the details of the use of MemCB to evaluate the performance and dependability of a Memcached service.

In the following, we demonstrate how the proposed architecture helps to reduce the cost and effort in building a benchmark for a new cloud computing service. We consider various dimensions for evaluation: cost of software development; design complexity; usability and adaptability. Cost of software development primarily includes primarily the coding effort to develop a performance and dependability benchmark. Design complexity consists of how many components are generic (that can be easily extended from the generic

architecture) and how many components need to be designed from scratch. Usability comprises the ease of using the benchmark in evaluating performance and dependability of real world cloud computing service. Adaptability suggests how easy it is to add a new feature in the benchmark. Example of a feature could be the addition of a new fault type.

3.5.1 Reduced development costs

This section includes the software development effort (focused on code reuse) needed to develop a new dependability and performance benchmark for the previous use cases following the architecture proposed in this chapter. Obviously, there is a part of the dependability and performance benchmark that depends on the workload to be injected, application dependant, that will require more or less effort based on the semantics of the application. We focus thus on the effort needed to inject the different types of faults. Of course, this will depend again on the number of different faults that are considered to evaluate the dependability of a specific application. Table 3.1 shows the details of the software prototypes developed for performance and dependability benchmarking of MepReduce and Memcached i.e. MRBS and MemCB respectively. For both prototypes, we provide the details of total lines of code and performance and dependability specific components.

Table 3.1: Comparison of effort to build a new benchmark

Evaluation Parameter	**MRBS**	**MemCB**
Total lines of code	10,606	454
Performance Application dependant	1,183	104
Performance Application independent	6,394	0
Dependability Application dependant	979	41
Dependability Application independent	673	0
# lines per fault injector	35	3

The MRBS benchmark was built using 10,606 lines of code. Among them, 1,183 correspond to the platform independent modules for workload generation, 6,394 correspond to the platform dependant modules for workload generation, 979 for the generic part of dependability benchmarking, 673 for the platform specific dependability benchmark and 1,377 for statistics generation. In the platform specific dependability benchmark, 535 lines correspond to the generation of faultloads from previous execution traces, and 138 to the fault type definitions and their injectors (80 lines to inject *Task Software Fault* and *Hanging Task* faults, and 58 to inject *Node Crash* and *Task Process Crash* faults).

The MemCB benchmark was build using 454 lines of code. This benchmark also includes various third party libraries. Among the total lines of code, 104 correspond to the platform dependent modules for workload generation, 41 for the platform specific dependability benchmark and 309 for statistics generation. In the platform specific dependability benchmark, 41 lines correspond to inject *Node Crash* and *Network Fault*.

3.5.2 Better usability

The architecture possesses better usability from the point of view of benchmark designers and also benchmark users. Due to standard design of components, it is always easier to develop a new benchmark. The use of configuration files for tuning the parameters makes it easier to use and and run the benchmark. The benchmark outputs such as graphs and HTML files provides an easier way to visualize the performance and dependability metrics. Most of the code of benchmark prototypes such as MRBS is in Java which makes it robust and platform independent.

3.5.3 Higher adaptability

The architecture is also flexible and adaptable. The addition of new features such as new fault types is not difficult. This can be done without modifying the generic components of the architecture. The addition of new workloads and dataloads also do not require major modifications to the existing code. The interfaces for faultload addition, injection, and workload injection are flexible.

3.6 Summary

For a benchmark suite to enable a thorough analysis of the dependability and performance of a cloud service, it must provide the following. First, it must enable *automatic faultload generation and injection* in cloud service. This should cover different fault types, injected at different rates, which will provide a means to analyze the effectiveness of fault-tolerance in a variety of scenarios. Second, it must allow to *quantify dependability levels* provided by cloud service's fault-tolerance systems, through an empirical evaluation of the availability and reliability of such systems, in addition to performance and cost metrics. Third, it must cover a *variety of application domains, workload and dataload characteristics*, ranging from compute-oriented to data-oriented applications, batch applications to online interactive applications. Finally,

the benchmark suite must be *portable and easy to use on a wide range of platforms*, covering different cloud infrastructures.

In this chapter, we have introduced the generic architecture to build performance and dependability benchmarks for cloud services. We described various components and modules responsible for injecting faults in cloud services in addition to the components responsible for measuring the performance and dependability. Subsequent chapters present the details of the use of the generic architecture to build two software prototypes: MRBS and MemCB, to benchmark two popular cloud services: MapReduce and Memcached.

Chapter 4

MRBS: Dependability & Performance Benchmarking for MapReduce

Contents

4.1 Introduction **58**

4.2 Background **60**

4.2.1 MapReduce 60

4.2.2 Hadoop MapReduce 60

4.2.3 Fault Tolerance in Hadoop 61

4.3 Overview of MRBS **62**

4.3.1 MRBS Benchmark Suite 66

4.3.2 MRBS Workload 68

4.3.3 MRBS Dataload 69

4.3.4 MRBS Faultload 71

4.3.5 MRBS Faultload Builder 72

4.3.6 MRBS Fault Injection 72

4.3.7 Performance and Dependability Analysis in MRBS . . 74

4.4 On the Portability of MRBS Software **76**

4.4.1 Portability of Fault Builder 76

4.4.2 Portability of Fault Injection 77

4.4.3 Portability of Performance and Dependability Analysis 77

4.4.4 Portability of Workload and Dataload Injection 77

4.4.5 Portability of MRBS Experiment Deployer 77

4.5 Experimental Evaluation **78**

4.5.1 Experimental Setup 78

4.5.2 Performance Evaluation Under Workloads 78

4.5.3 Dependability Evaluation Under Faultloads 81
4.6 Summary 83

4.1 Introduction

Cloud computing is increasingly being used, enabling on-demand services hosted on a virtually unlimited set of computing and storage resources. MapReduce has become a popular Big Data cloud service [Dean and Ghemawat, 2004], used by a wide range of applications such as log analysis, data mining, scientific computing [Chen and Schlosser, 2008], bioinformatics [Schatz, 2009], decision support and business intelligence [Gre, 2011].

MapReduce provides a convenient means for distributed data processing and automatic parallel execution on clusters of machines. It has various applications and is used by several services featuring fault-tolerance and scalability. MapReduce offers developers a means to transparently handle data partitioning, replication, task scheduling and fault-tolerance on a cluster of commodity computers. Hadoop, one of the most popular MapReduce frameworks, provides key fault-tolerance features such as handling node failures, task failures, hanging tasks [White, 2009].

There has been a considerable interest in improving fault-tolerance and performance of MapReduce. Several efforts have explored on-demand fault-tolerance [Fadika and Govindaraju, 2010], replication and partitioning policies [Ananthanarayanan et al., 2011], [Eltabakh et al., 2011], adaptive fault-tolerance [Jin et al., 2012, Lin et al., 2010], extending MapReduce with other fault-tolerance models [Costa et al., 2011, Ko et al., 2010]. There are works that focus upon task scheduling policies [Isard et al., 2009, Zaharia et al., 2010, Zaharia et al., 2008], resource provisioning [Verma et al., 2011] and cost-based optimization techniques [Herodotou and Babu, 2011]. There has also been a considerable interest in extending MapReduce with other fault tolerance models [Bessani et al., 2010], or with techniques from database systems [Lu, 2010, Abouzeid et al., 2009, Dittrich et al., 2010, Floratou et al., 2011].

However, there has been very little in the way of empirical evaluation of MapReduce dependability, and the impact of failures on performance. Evaluations have often been conducted in an ad-hoc manner, such as turning off a node in the MapReduce cluster or killing a task process. These actions are typically dictated by what testers can actually control, but may lead to low coverage testing. Recent tools, like Hadoop fault injection framework [Had, 2011b], offer the ability to emulate non-deterministic exceptions in the HDFS distributed filesystem underlying Hadoop MapReduce. Although they provide a means to program unit tests for HDFS, such low-level tools are meant to be used by developers who are familiar with the internals of HDFS, and are unlikely to be used by end-users of MapReduce systems.

MapReduce fault injection must therefore be generalized and automated

for higher-level and easier use. Not only it is necessary to automate the injection of faults, but also the definition and generation of MapReduce faultloads. A faultload will describe *what* fault to inject (e.g. a node crash), *where* to inject it (e.g. which node of the MapReduce cluster), and *when* to inject it (e.g. five minutes after the application started). Furthermore, most evaluations of MapReduce fault-tolerance systems relied on microbenchmarks based on simple MapReduce programs and workloads, such as *grep*, *sort* or *word count*. While microbenchmarks may be useful in targeting specific system features, they are not representative of full distributed applications, and they do not provide multi-user realistic workloads.

These observations motivate the design of MRBS (MapReduce Benchmark Suite), the first benchmark suite for evaluating the dependability *and* performance of MapReduce systems.

The contributions of this chapter are following:

- We provide automatic faultload generation and injection in MapReduce. This covers different fault types, injected at different rates, which will provide a means to analyze the effectiveness of fault-tolerance in a variety of scenarios.

- We provide a benchmark suite that covers five application domains: recommendation systems, business intelligence, bioinformatics, text processing, and data mining. It supports a variety of workload and dataload characteristics, ranging from compute-oriented to data-oriented applications, batch applications to online interactive applications. Indeed, while MapReduce frameworks were originally limited to offline batch applications, recent works are exploring the extension of MapReduce beyond batch processing [Condie et al., 2010], [Liu and Orban, 2011]. The proposed benchmark suite uses various input data sets from real applications, among which an online movie recommender service [Mov, 2011], Wikipedia [Wik, 2012], and real genomes for DNA sequencing [San, 2011].

- We describe the design principles of MRBS benchmark suite, and its deployment on Hadoop clusters running on Amazon EC2, and on a private cloud. Although the current MRBS prototype is provided for the Hadoop popular MapReduce framework, we believe that the proposed dependability and performance benchmarking solution can be easily applied to other MapReduce frameworks. We, thus, discuss the portability of MRBS to other MapReduce frameworks.

- We describe how MRBS allows automatic deployment of experiments on cloud infrastructures. It does not depend on any particular infras-

tructure and can run on different private or public clouds. This makes dependability and performance benchmarking easy to adopt by end-users of MapReduce, and by developers of MapReduce fault-tolerance and scalability solutions.

- MRBS is available as a software prototype to help researchers and practitioners to better analyze and evaluate the dependability and performance of MapReduce systems; it can be downloaded from: `http://sardes.inrialpes.fr/research/mrbs`.

4.2 Background

4.2.1 MapReduce

MapReduce is a programming model and a software framework introduced by Google in 2004 to support distributed computing and large data processing on clusters of commodity machines [Dean and Ghemawat, 2004]. The MapReduce functional programming model provides a simple means to write programs that process large input data sets. Programmers write only two main functions: a *map* function and a *reduce* function, and the MapReduce framework automatically handles data and computation distribution in a cluster. A MapReduce *job*, i.e. an instance of a running MapReduce program, has several phases; each phase consists of multiple *tasks* scheduled by the MapReduce framework to run in parallel on cluster nodes. First, input data are divided into *splits*, one split is assigned to each map task. During the mapping phase, tasks execute a *map* function to process the assigned splits and generate intermediate output data. Then, the reducing phase runs tasks that execute a *reduce* function to process intermediate data and produce the output.

4.2.2 Hadoop MapReduce

There are many implementations of MapReduce. Hadoop is a popular MapReduce framework, available in public clouds such as Amazon, and Open Cirrus [Ama, 2011b, Ope, 2011a]. Hadoop cluster consists of a *master node* and *slave nodes*. Users (i.e. clients) of a Hadoop cluster submit MapReduce jobs to the master node which hosts the *JobTracker* daemon that is responsible of job scheduling. By default, jobs are scheduled in FIFO mode and each job uses the whole cluster until the job completes, or until the job finishes its map phase in which case map slots are available for another job. However, other multi-user job scheduling approaches are also available in Hadoop to

allow jobs to run concurrently on the same cluster. This is the case of the *fair scheduler* which assigns every job a fair share of the cluster capacity over time [Fai, 2011]. Moreover, each slave node hosts a *TaskTracker* daemon that periodically communicates with the master node to indicate whether the slave is ready to run new tasks. If it is, the master schedules appropriate tasks on the slave. Each task is executed by a separate process.

Hadoop framework also provides a distributed filesystem (HDFS) that stores data across cluster nodes. HDFS architecture consists of a *NameNode* and *DataNodes*. The *NameNode* daemon runs on the master node and is responsible of managing the filesystem namespace and regulating access to files. A *DataNode* daemon runs on a slave node and is responsible of managing storage attached to that node. HDFS is thus a means to store input, intermediate and output data of Hadoop MapReduce jobs. In addition, for fault tolerance purposes, HDFS replicates data on different nodes.

4.2.3 Fault Tolerance in Hadoop

Hadoop is able to tolerate failures of different types [White, 2009], as described in the following.

Node Crash. In case of a slave node failure, the *JobTracker* on the master node stops receiving heartbeats from the *TaskTracker* on the slave for an interval of time. When it notices the failure of a slave node, the master removes the node from its pool and reschedules tasks that were ongoing on other nodes. The heartbeat timeout is set in the *mapred.task.tracker.expiry.interval* Hadoop property. In the current implementation of Hadoop, failures of the master node are not tolerated.

Task Process Crash. A task may also fail because a map or reduce task process suddenly crashes, e.g., due to a transient bug in the underlying (virtual) machine. Here again, the parent *TaskTracker* notices that a task process has exited and notifies the *JobTracker* for possible task retries.

Task Software Fault. A task may fail due to errors and runtime exceptions in *map* or *reduce* functions written by the programmer. When a *TaskTracker* on a slave node notices that a task it hosts has failed, it notifies the *JobTracker* on the master node which reschedules another execution of the task, up to a maximum number of retries. This allows to tolerate transient errors in MapReduce programs.

Hanging Tasks. A map or reduce task is marked as failed if it stops sending progress updates to its parent *TaskTracker* for a period of time (indicated by *mapred.task.timeout* Hadoop property). If that occurs, the task process is killed, and the *JobTracker* is notified for possible task retries.

4.3 Overview of MRBS

MRBS is a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems. As discussed in Chapter 3, MRBS achieve the following design objectives:

1. **Multi-criteria analysis.** MRBS aims to measure and analyze the performance *and* dependability of MapReduce systems. In particular, we consider several measurement metrics such as reliability, availability, financial cost, request response time (i.e. latency), and request throughput. We also consider low-level MapReduce metrics, such as throughput of MapReduce jobs and tasks, task and job failures, I/O throughput (data reads/writes), etc.

2. **Diversity.** MRBS covers a variety of application domains and programs with a wide range of MapReduce characteristics. This includes data-oriented applications vs. compute-oriented applications. Furthermore, whereas MapReduce was originally used for long running batch jobs, modern MapReduce cluster is shared between multiple users running concurrently [Condie et al., 2010, Liu and Orban, 2011]. Therefore, MRBS considers batch applications as well as interactive applications. Moreover, MRBS allows to characterize different aspects of application load such as the *faultload*, the *workload* and the *dataload*. Roughly speaking, the faultload describes MapReduce fault types and fault arrival rates. The workload is characterized by the number of clients (i.e. users) sharing a MapReduce cluster, the types of client requests (i.e. MapReduce programs), and request arrival rates. The dataload characterizes the size and nature of MapReduce input data.

3. **Usability.** MRBS is easy to use, configure and deploy on a MapReduce cluster. It is independent from any infrastructure and can easily run on different public clouds and private clouds. MRBS provides results which can be readily interpreted in the form of monitored statistics and automatically generated charts.

MRBS allows to inject various faultloads, workloads and dataloads in MapReduce systems, and to collect information that helps testers understand the observed behavior of MapReduce systems. The overall architecture of MRBS is presented in Figure 4.1. MRBS comes with a benchmark suite, that is a set of five benchmarks covering various application domains: recommendation systems, business intelligence, bioinformatics, text processing,

and data mining. Conceptually, each benchmark implements a service running on a MapReduce cluster, and each service has several types of requests that are issued by users (i.e. clients). A client request executes one or a series of MapReduce jobs. MRBS may emulate multiple clients implemented as external entities, that concurrently access the MapReduce cluster.

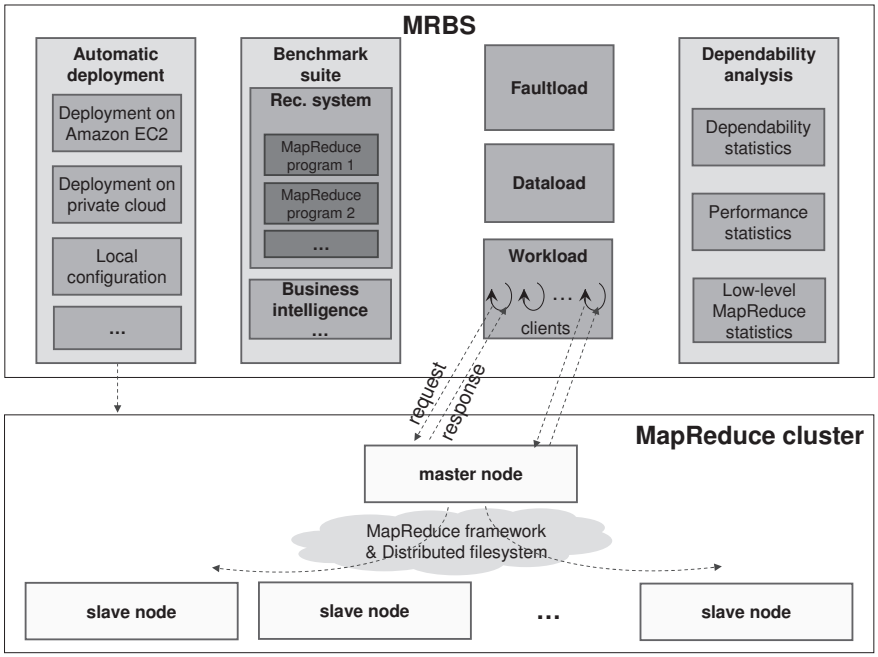
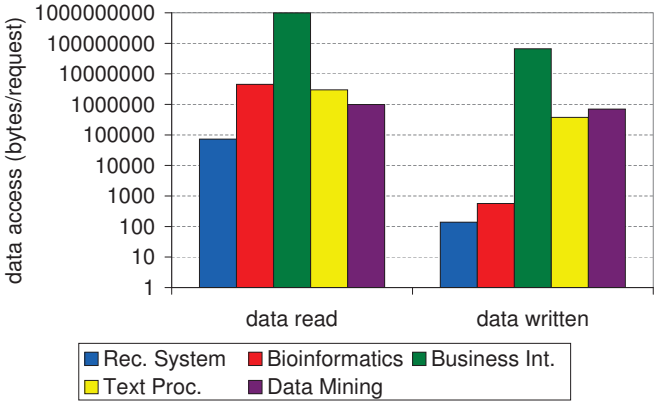


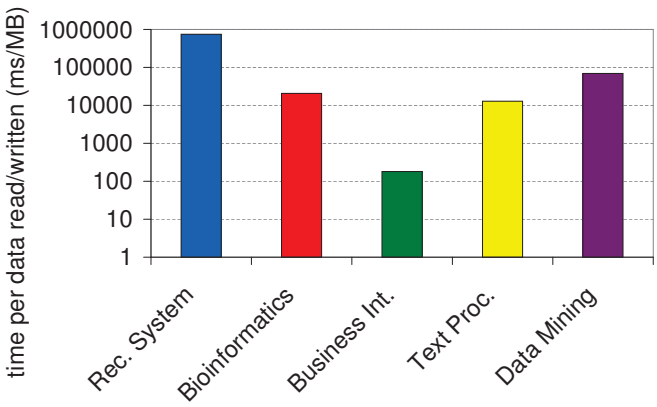
Figure 4.1: Overview of MRBS

MRBS benchmarks were chosen to exhibit different behaviours in terms of computation pattern and data access pattern: the Recommendation System is a compute-intensive benchmark, the Business Intelligence system is a data-intensive benchmark, and the other benchmarks are relatively less compute/data-intensive. This is shown in Figure 4.2 that compares the different benchmarks (note the logarithmic scale). Figure 4.2(a) gives the average size of data accessed per client request, and Figure 4.2(b) gives the processing time per unit of accessed data. Moreover, Figure 4.3 shows that MRBS benchmarks present different MapReduce characteristics in terms of the average number of MapReduce jobs and tasks per client request. Figure 4.3(a) gives the average MapReduce tasks per job, and Figure 4.3(b) gives the average MapReduce tasks per client request.

[0]figures obtained with the first dataload of each benchmark

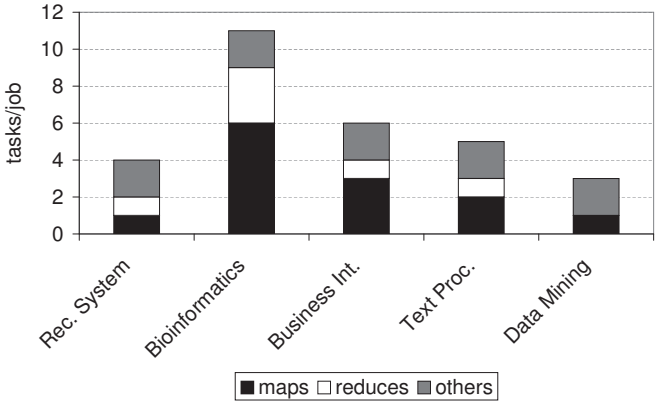


(a) Data access per client request

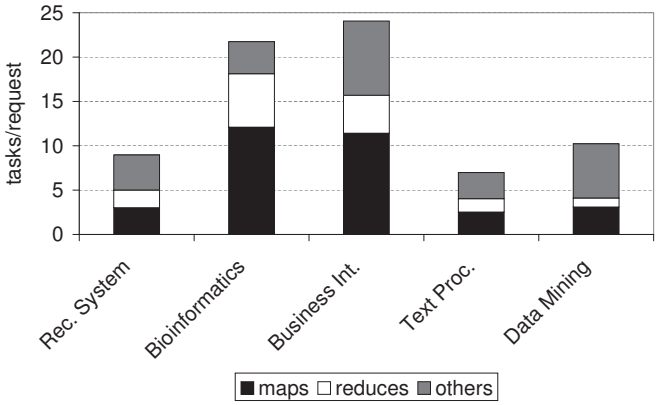


(b) Processing time per data size

Figure 4.2: Data-oriented vs. compute-oriented benchmarks



(a) MapReduce tasks per job



(b) MapReduce tasks per client request

Figure 4.3: MapReduce characteristics of benchmarks

4.3.1 MRBS Benchmark Suite

Table 4.1: Workloads and applications constituting MRBS

Workload Category	**Application**	**Description**
Recommendation System	Recommendation Based upon Users Recommendation Based upon Items	finds movies a user might like or movies similar to another movie
Business intelligence	Complex HIVE Queries	performs complex business analysis
Bioinformatics	CloudBurst DNA sequence analysis algorithm	maps sequence data to reference genomes
Text Processing	Grep Sort WordCount	searches text files for lines containing a match to a given text pattern sorts the content of text files counts the number of words in text files
Data Mining	K Means Clustering Baysian Classification	groups the contents of text files together into clusters decides what are the documents belonging to newsgroups

Benchmarks constituting MRBS are summarized in Table 4.1. We present technical details of the benchmarks constituting MRBS in the following.

Recommendation System. Recommendation systems are widely used in e-commerce sites such as Amazon.com which, based on purchases and site activity, recommends books likely to be of interest. MRBS implements an online movie recommender system. It builds upon a set of movies, a set of users, and a set of ratings and reviews users give for movies to indicate whether and how much they liked or disliked the movies. These data have been collected from a real movie recommendation web site [Mov, 2011]. The benchmark provides four types of operations. First, a user may ask for all ratings and reviews given by other users for a given movie, to see whether people liked or disliked the movie. The recommendation system also allows to browse all ratings and reviews given by a user. Furthermore, a user may ask the recommender system

to provide him/her the top ten recommendations, these are the movies this user would like the most. Another type of operation a user may perform is to ask the recommendation system how it would recommend him/her a given movie; this would indicate to the user whether and how much he/she would like or dislike that movie.

This benchmark is based on MapReduce implementations of data mining and search algorithms. For building recommendations, similarities between movies are computed by looking to users' ratings and preferences. The algorithm uses item-based recommendation techniques to find movies that are similar to other movies [Jannach et al., 2010]. Similarities between movies are relatively static and can thus be computed once and then reused. This is what the Recommendation system benchmark does by storing the precomputed similarities in the distributed filesystem. Therefore, the benchmark handles client requests by applying a search algorithm on the precomputed data.

Business Intelligence. The Business Intelligence benchmark represents a decision support system for a wholesale supplier. It implements business-oriented queries that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions. MRBS includes a MapReduce implementation of the TPC-H industry-standard benchmark [TPC, 2011a]. It uses Apache Hive on top of Hadoop, a data warehouse that facilitates ad-hoc queries using a SQL-like language called HiveQL [Hiv, 2013]. The benchmark consists of eight data tables, and provides 22 types of operations among which an operation that identifies geographies where there are customers who may be likely to make a purchase, or an operation that retrieves the ten unshipped orders with the highest value. The provided operations are implemented as HiveQL queries, that are translated into MapReduce programs by the Hive framework. The input data of the benchmark were generated with the DBGen TPC provided software package, and are compliant with the TPC-H specification [TPC, 2011a].

Bioinformatics. The Bioinformatics benchmark performs DNA sequencing. Users of the benchmark may choose a complete genome to analyze among a set of genomes. Roughly speaking, DNA sequencing attempts to find where reference reads (i.e. short DNA sequences) occur in a genome, allowing a fixed number of errors. This is a highly parallelizable process that can benefit from MapReduce. The benchmark includes a MapReduce-based implementation of DNA sequencing [Schatz, 2009]. The data used in the benchmark are publicly available genomes [San, 2011]. Currently, the benchmark allows to analyze several genomes of organisms such as the pathogenic organisms Salmonella Typhi, Rhodococcus equi, and Streptococcus suis. The Salmonella Typhi is

a parasite that causes human typhoid fever; its genome is about 2,000,000 DNA characters long. The Rhodococcus equi is a disease-causing organism in horses, with a genome of about 3,000,000 DNA characters. Streptococcus suis is another pathogen which human infection can cause severe outcomes such as meningitis; its genome is also about 2,000,000 DNA characters long. The benchmark can be easily extended with new genomes to analyze by simply defining them as new input data in MRBS configuration file.

Text Processing. Text processing is a classical application of MapReduce used, for instance, to analyze the logs of web sites and search engines. MRBS provides a MapReduce text processing-oriented benchmark, with three types of operations allowing clients to search words or word patterns in text documents, to know how often words occur in text documents, or to sort the contents of documents. The benchmark uses synthetic input data that consist of randomly generated text files of different sizes.

Data Mining. This benchmark provides two types of data mining operations: clustering and classification [Mah, 2013]. Bayesian classification assigns a category from a fixed set of known categories to an un-categorized element. As an example of classification application, Yahoo! Mail classifies incoming messages as spam, or not, based on prior emails and spam reports. MRBS benchmark considers the case of classifying newsgroup documents into categories. A first step consists in applying a learning algorithm to train the model. Then, the model can be used with an un-classified document to estimate the newsgroup the document is likely to belong to. The benchmark uses collections of data publicly available from [20N, 2011].

Furthermore, the benchmark provides canopy clustering operations. Canopy clustering partitions a large number of elements into clusters in such a way that elements belonging to the same cluster share some similarity. For instance, Google News uses clustering techniques to group news articles according to their topic. The benchmark uses datasets of synthetically generated control charts, to cluster the charts into different classes based on their characteristics [Mah, 2013].

4.3.2 MRBS Workload

The workload is characterized by the benchmark to execute, and the number of concurrent clients issuing requests on that benchmark application; this number may vary. The workload is also characterized by the execution mode which may be interactive or batch. In interactive mode, concurrent clients share the MapReduce cluster (i.e. have their requests executed) at the same time. In batch mode, requests from different clients are executed in FIFO order. In

interactive mode, clients concurrently and fairly share the MapReduce cluster (see Section 4.2.2).

The workload is also characterized by client request distribution, that is the relative frequencies of different request types. It may follow different distribution laws (known as workload mixes), such as a random distribution. Request distribution may be defined using a state-transition matrix that gives the probability of transitioning from one request type to another.

4.3.3 MRBS Dataload

The dataload is characterized by the nature and size of data sets used as inputs for a benchmark. Data used in MRBS' benchmarks are real and publicly available datasets. Users of MRBS may choose between datasets of different sizes (see Table 4.2, the default input data for each benchmark being the first one). Obviously, the nature and format of data depend on the actual benchmark. In the following, we describe the nature of data used by the different benchmarks.

The *Recommendation System* benchmark uses a dataset composed of four subsets: a set of movies, a set of users, a set of ratings given by the users for movies, and the estimated recommendation of all the movies for each user. The first three sets have been collected from a real movie recommendation web site [Mov, 2011]. The fourth is generated with a preprocessing of the first datasets [Mah, 2013, Jannach et al., 2010].

The *Business Intelligence* benchmark uses a database that consists of eight data tables. The database is generated with TPC-H's DBGen database population tool [TPC, 2011a].

The *Bioinformatics* benchmark's dataset is the genome of several real pathogenic organisms, namely *Salmonella typhi*, *Rhodococcus equi*, and *Streptococcus suis*, which are publicly available [San, 2011].

The *Text Processing* benchmark uses as input data the Wikipedia content in different languages, e.g. English, Italian etc. [Wik, 2012].

The *Data Mining* benchmark uses publicly available data sets from [Mah, 2013] and [20N, 2011]. The first one is used for clustering requests, the second is used for classification requests.

MRBS can be used for benchmarking the dependability of MapReduce, for benchmarking the performance of MapReduce, or for both. In the following, we present dependability benchmarking; performance benchmarking is then discussed in Section 4.3.7. To use MRBS for dependability benchmarking, three main steps are needed: (i) build a faultload (i.e. fault scenario) to describe the set of faults to be injected, (ii) conduct fault injection experiments

Table 4.2: Application domains and benchmark characteristics in MRBS. An appended '+' symbol indicates larger dataload, higher computation contention or higher data access contention.

Domain	**Dataload**	**Execution mode**	**Workload**	**Faultload**	**Computation vs. data access**
Recommendation system	dataload 100,000 ratings, 1000 users, 1700 movies dataload+ 1 million ratings, 6000 users, 4000 movies dataload++ 10 million ratings, 72,000 , 10,000 movies	interactive / batch	mono-user / multi-user	no fault / faults	compute-oriented+
Business intelligence	dataload 1GB dataload+ 10GB dataload++ 100GB dataload* any data size	interactive / batch	mono-user / multi-user	no fault / faults	data-oriented+
Bioinformatics	dataload genomes of 2,000,000 to 3,000,000 DNA characters	interactive / batch	mono-user / multi-user	no fault / faults	data-oriented / compute-oriented
Text processing	dataload text files (1GB) dataload+ text files (10GB) dataload++ text files (100GB) dataload* any data size	interactive / batch	mono-user / multi-user	no fault / faults	data-oriented / compute-oriented
Data mining	dataload 5000 documents, 5 newsgroups, 600 control charts dataload+ 10,000 documents, 10 newsgroups, 1200 control charts dataload++ 20,000 documents, 20 newsgroups, 2400 control charts	interactive / batch	mono-user / multi-user	no fault / faults	data-oriented / compute-oriented

based on the faultload, and (iii) collect statistics about dependability levels of the MapReduce system under test. This is presented in Figure 4.4.

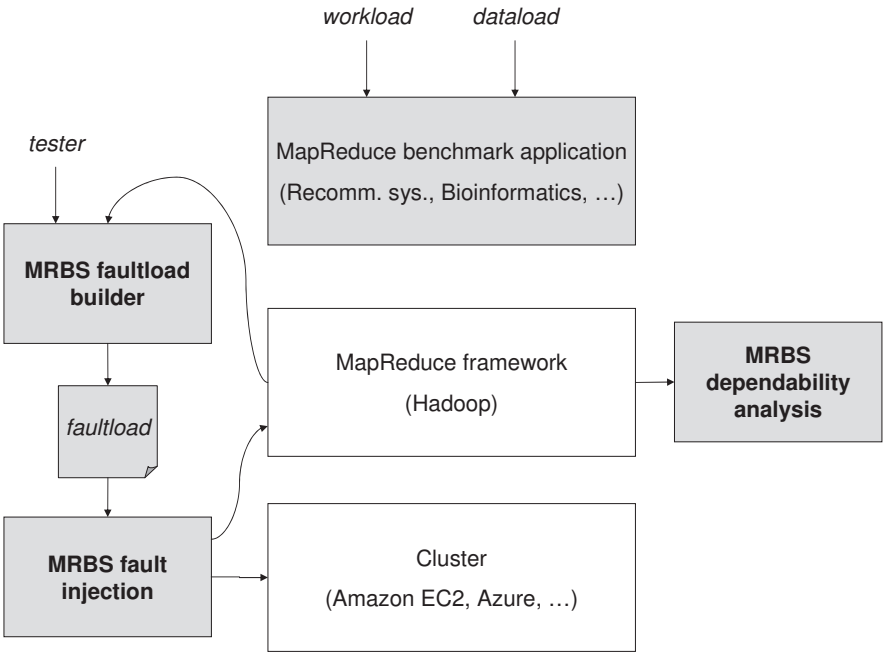


Figure 4.4: Overview of MRBS dependability benchmarking

The evaluator of the dependability of a MapReduce system chooses an application from MRBS ' set of benchmarks, depending on the desired application domain and whether he/she targets compute-oriented or data-oriented applications. MRBS injects workload and dataload in the system under test. MRBS also allows the evaluator to choose specific dataload and workload, to stress the scalability of the MapReduce system. Further details about workload and dataload injection, and performance analysis with MRBS are given in Section 4.3.7.

4.3.4 MRBS Faultload

A faultload in MRBS is described in a file, either by extension, or by intention. In the former case, each line of the faultload file consists of the following elements: the time at which a fault occurs (relatively to the beginning of the experiment), the type of fault that occurs and, optionally, where the fault occurs. A fault belongs to one of the fault types handled by Hadoop MapReduce, and introduced in Section 4.2.3. A fault occurs in one of the MapReduce

cluster nodes; this node may be either explicitly specified in the faultload or randomly chosen among the set of nodes. To make the parsing of this faultload file more efficient, redundant lines, that correspond to multiple occurrences of the same fault at the same time, are grouped into one line with an extra parameter that represents the number of occurrences of that fault. Another way to define a more concise faultload is to describe it by intention. Here, each line of the faultload file consists of: a fault type, a fault distribution function and the mean time between failures (MTBF).

4.3.5 MRBS Faultload Builder

Testers can explicitly build synthetic faultloads representing various fault scenarios. A faultload description may also be automatically obtained, either randomly or based on previous application runs' traces. Random faultload builder may produce a faultload by extension or by intention. In the case of faultload by extension, it generates the i-th line of the faultload file as follows: $< \ time_stamp_i, \ fault_type_i, \ fault_location_i \ >$, with $time_stamp_i$ being a random value between $time_stamp_{i-1}$ (or 0 if $i = 1$) and the length of the experiment, $fault_type_i$ and $fault_location_i$ random values in the set of possible values. However, in case of faultload by intention, random faultload builder produces a faultload description where, with each fault type is associated a random MTBF between 0 and the length of the experiment.

A faultload description may also be automatically generated based on traces of previous runs of MapReduce applications and workloads. The trace-based faultload builder parses the MapReduce framework's logs and identifies the faults that occurred in these runs: their time stamp, their type, and possibly their location. We designed the trace-based faultload builder to work directly on the MapReduce framework's logs, which allows it to work not only on workloads and benchmark applications from the MRBS benchmark suite, but also with other workloads and MapReduce applications. As with the other variants of the faultload builder, the faultload that results from the trace-based faultload builder may be described by extension or intention. In the latter case, a statistical analysis of the traces is performed to calculate MTBF for the different types of faults.

4.3.6 MRBS Fault Injection

The output of the MRBS faultload builder is passed to the MRBS fault injector. The MRBS fault injector divides the input faultload into subsets of faultloads as follows: one global faultload that groups all crash faults that

will occur in all nodes of the MapReduce cluster (i.e. node crash, task process crash), and per-node faultloads that group all occurrences of other types of faults that will occur in each node (i.e. task software faults, hanging tasks).

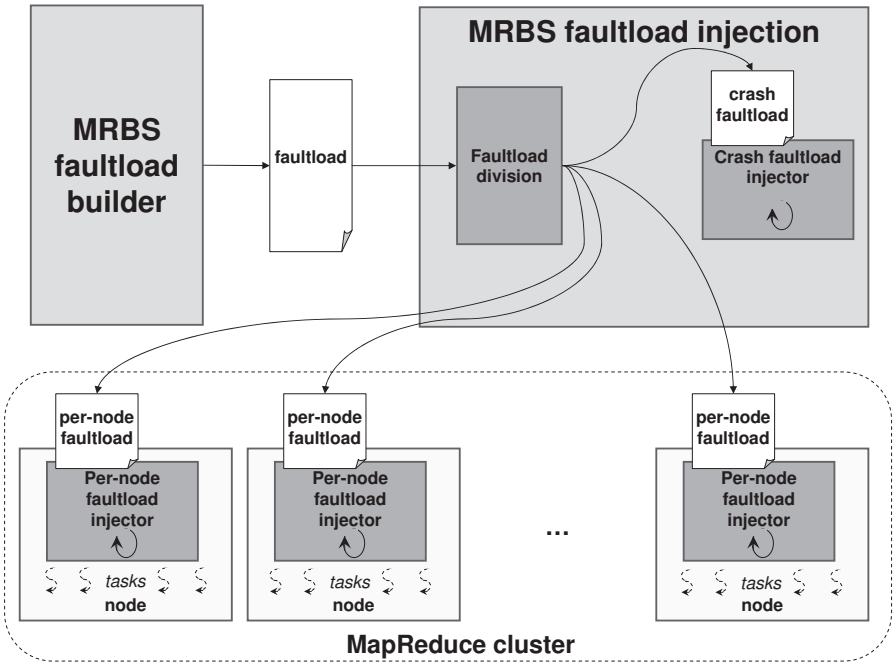


Figure 4.5: Architecture of MRBS faultload injector

The MRBS fault injector runs a daemon that is responsible of injecting the global faultload. In the following, we present how the daemon injects these faults, in case of a faultload described by extension, although this can be easily generalized to a faultload described by intention. Thus, for the i-th fault in the crash faultload, the dameon waits until $time_stamp_i$ is reached, then calls the fault injector of $fault_type_i$ (see below), on the MapReduce cluster node corresponding to $fault_location_i$. This fault injector is called as many times as there are occurrences of the same fault at the same time. The fault injection dameon repeats these operations for the following crash faults, until the end of the faultload file is encountered or the end of the experiment is reached.

The MRBS fault injector handles the per-node faultloads differently. A per-node faultload includes faults that occur inside tasks. MRBS intercepts task creation to check whether a fault must be injected in that task, in which case the fault injector corresponding to the fault type is called (see below). MRBS does not require the modification of the source code of the MapReduce

framework. Instead, it synthesizes a new version of the MapReduce framework library using aspect-oriented techniques. The synthetic MapReduce library has the same API as the original one, but underneath this new library includes task creation interceptors that encode the fault injection logic. The overall architecture of the faultload injection in MRBS is described in Figure 4.5.

Node Crash Injection. A node crash is simply implemented by shutting down a node. This fault injector uses the API of the underlying cloud infrastructure to implement such a fault. For example, in case of a public cloud such as Amazon EC2, a node crash consists in a premature termination of an Amazon EC2 instance. However, if a tester wants to conduct multiple runs of the same dependability experiment, and if faults are implemented by shutting down machines, new machines must be acquired from the cloud at the beginning of each run, which may induce a delay. For efficiency purposes, we propose an implementation of MapReduce node fault which kills all MapReduce daemons running on that node. Specifically, in the case of Hadoop these include the *TaskTracker* and *DataNode* daemons running in a slave node[1]. The timeout to detect a MapReduce node failure is set to 30 seconds, a value set in *mapred.task.tracker.expiry.interval* Hadoop property.

Task Process Crash Injection. This type of fault is implemented by killing the process running a task on a MapReduce node.

Task Software Fault Injection. A task software fault is implemented as a runtime exception thrown by a map task or a reduce task. This fault injector is called by the interceptors injected into the MapReduce framework library by MRBS .

Provoking Hanging Tasks. A task is marked as hanging if it stops sending progress updates for a period of time. This type of fault is injected into a map task or a reduce task through the interceptors that make the task sleep a longer time than the maximum period of time for sending progress updates (*mapred.task.timeout* Hadoop property).

4.3.7 Performance and Dependability Analysis in MRBS

Performance and Dependability Statistics

MRBS can be used to evaluate the performance and/or the dependability of MapReduce systems, using various workloads, dataloads, and faultloads. MRBS produces runtime statistics related to dependability, such as reliability, and availability [Laprie, 1995]. Reliability is measured as the ratio of

[1]A node crash is not injected to the MapReduce master node since this node is not fault-tolerant in the used version of Hadoop, c.f. Section 4.2.2.

successful MapReduce client requests to the total number of requests, during a period of time. Availability is measured from the client's perspective as the ratio of, on the one hand, the time the benchmark service is capable of returning successful responses to the client, and on the other hand, the total time; availability is measured during a period of time.

In addition, MRBS produces performance and cost statistics, such as client request response time, request throughput, and the financial cost of a client request. Throughput is the number of clients requests handled by the benchmark per unit of time. Response time is the elapsed time from the moment the client submits a request until the response is received by the client. This may include, not only the execution time of that request, but also the overhead of time-sharing in Hadoop cluster.

MRBS also provides low-level MapReduce statistics related to the number, length and status (i.e. success or failure) of MapReduce jobs, map tasks, reduce tasks, the size of data read from or written to the distributed file system, etc. These low-level statistics are built offline, after the execution of the benchmark. Optionally, MRBS can generate charts plotting continuous-time results.

Automatic Deployment of Experiments

MRBS allows to automatically deploy extensive experiments and test various scenarios on cloud infrastructures such as Amazon EC2 and private clouds. The cloud infrastructure and the size of the cluster are configuration parameters of MRBS. MRBS acquires on-demand resources provided by cloud computing infrastructures such as private clouds, or the Amazon EC2 public cloud [Ama, 2011a]: one node is dedicated to run MRBS load injectors, and the other nodes are used to host the MapReduce cluster. MRBS automatically releases the resources when the benchmark terminates. We expect to provide MRBS versions for other cloud infrastructures such as the OpenStack open source cloud infrastructure [Ope, 2011b]. Once the cluster is set up, the MapReduce framework and its underlying distributed file system are started on the cluster. The current implementation of MRBS uses the popular Apache Hadoop MapReduce framework and HDFS.

Usability of MRBS

Once the user of MRBS has defined a workload, a dataload, and a faultload (or used the default ones), MRBS automatically deploys the experiment on a cluster in specified cloud, and injects the load into the MapReduce cluster. It first uploads input data in the MapReduce distributed file system. This is

done once, at the beginning of the benchmark, and the data are then shared by all client requests. Afterwards, it creates as many threads as concurrent clients there are. Thread clients will remotely send requests to the master node of the MapReduce cluster which schedules MapReduce jobs in the cluster (see Figure 4.1). Clients continuously send requests/receive responses until the execution run terminates.

An experiment run has three successive phases: a warm-up phase, a run-time phase, and a slow-down phase. Statistics are produced during the run-time phase, whereas the warm-up phase allows the MapReduce system to reach a steady state before collecting statistics, and the slow-down phase allows to terminate the benchmark in a clean way. An experiment may also be automatically run a number of times, to produce average statistics and variance reports.

To make MRBS flexible, a configuration file is provided, that involves several parameters such as the length of the experiment, the size of MapReduce input data, etc. Nevertheless, to keep MRBS simple to use, these parameters come with default values that may be adjusted by MRBS users (See Annexe A.1).

4.4 On the Portability of MRBS Software

Although the current version of MRBS prototype is provided for Hadoop, we believe that the proposed dependability and performance benchmarking solution can be easily applied to other MapReduce frameworks. Most of MRBS prototype is general enough and applies to any MapReduce framework, except some specific parts. In the following, we describe how to port these parts to other MapReduce frameworks, discussing the different elements presented in Figure 4.4.

4.4.1 Portability of Fault Builder

The faultload builder in MRBS is general enough to not to rely on the underlying MapReduce framework, except the trace-based faultload generation. Indeed, to generate faultloads based on traces, the faultload builder analyzes the logs produced by the MapReduce framework, and uses pattern recognition to detect the occurrence of faults. This pattern recognition should, therefore, be adapted to a specfic MapReduce framework.

4.4.2 Portability of Fault Injection

Fault injection techniques used in MRBS depend on the actual type of faults to inject. For instance, the injection of task software faults or hanging tasks depends on the MapReduce API. This API is needed to capture the task creation point at which a fault will be injected by MRBS using aspect-oriented techniques. Thus, the injection of faults of these types needs to be adapted for any new MapReduce API. Furthermore, the injection of node crash faults or task process crash faults depends on the actual names of the underlying processes and, thus, needs to be adapted for a new MapReduce framework.

4.4.3 Portability of Performance and Dependability Analysis

High-level statistics such as response time, throughput, availability and reliability are computed by MRBS based on the MapReduce framework's output files. And low-level statistics are extracted from the the MapReduce framework's log files. In both cases, pattern recognition is applied to these files to extract the necessary information. Thus, this pattern recognition should be adapted to the framework in use.

4.4.4 Portability of Workload and Dataload Injection

Workload injection and dataload injection depend on the MapReduce framework user interface. The former uses this interface to send the request (i.e. MapReduce jobs) of the emulated clients to the MapReduce cluster. The latter uses this interface to upload the input data to the MapReduce filesystem. Porting MRBS' workload injection and dataload injection to a new MapReduce framework is, thus, straightforward.

4.4.5 Portability of MRBS Experiment Deployer

MRBS allows automatic deployment of experiments, and this is implemented mainly using the MapReduce framework user interface. This inludes operations such as starting and stopping the MapReduce service, copying the datasets to the framework filesystem, selecting the MapReduce job scheduler, etc. Moreover, to automate the deployment of experiments on a given cloud infrastructure (e.g. Amazon EC2), MRBS uses the cloud user interface to start/stop instances. Thus, MRBS' automatic deployment of experiments should be adapted to the user interface of the MapReduce framework in use, and the user interface of the target cloud infrastructure.

4.5 Experimental Evaluation

In the following, we present the use of MRBS to evaluate the performance and dependability of MapReduce systems.

4.5.1 Experimental Setup

The experiments presented in this section were conducted in a cluster running on Amazon EC2 [Ama, 2011a], and on Grid'5000 [Bolze et al., 2006]. Each cluster consists of one node hosting MRBS and emulating concurrent clients, and a set of nodes hosting the MapReduce cluster. The experiments below use several benchmarks of MRBS with default (unless specified otherwise) dataloads (see Table 4.2); the benchmarks are run in interactive mode, with multiple concurrent clients. In these experiments, client request distribution is random, and request interarrival time is an average of 7 seconds. Availability and reliability are measured in periods of 30 minutes, and cost is based on Amazon EC2 pricing at the time we conducted the experiments, which is $0.34 per instance·hour.

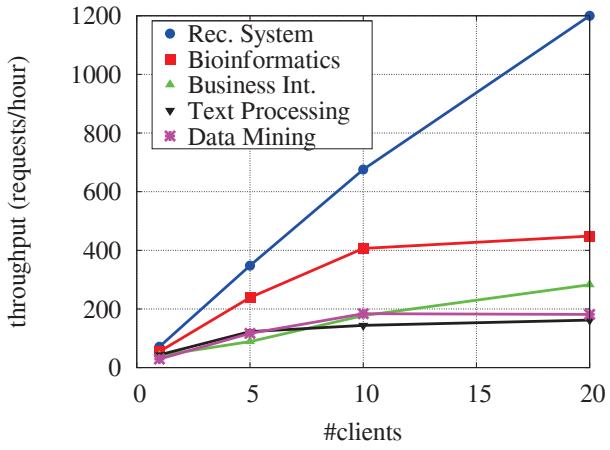
The hardware configuration used in the experiments is described in Table 4.3. The underlying software configuration is as follows. We used Amazon EC2 large instances which run Fedora Linux 8 with kernel v2.6.21. Nodes in Grid'5000 run Debian Linux 6 with kernel v2.6.32. The MapReduce framework is Apache Hadoop v0.20.2, and Hive v0.7, on Java 6. MRBS uses Apache Mahout v0.6 data mining library [Mah, 2013], and CloudBurst v1.1.0 DNA sequencing library [Schatz, 2009].

Table 4.3: Hardware configurations of MapReduce Clusters

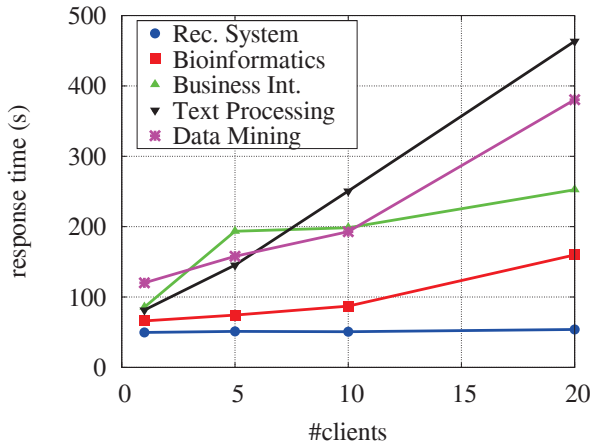
Cluster	CPU	Memory	Storage	Network
Amazon EC2	4 EC2 Compute Units in 2 virtual cores	7.5 GB	850 MB	10 Gbit Ethernet
Grid'5000	4-core 1-cpu 2.53 GHz Intel Xeon X3440	16 GB	278 GB SATA II	Infiniband 20G

4.5.2 Performance Evaluation Under Workloads

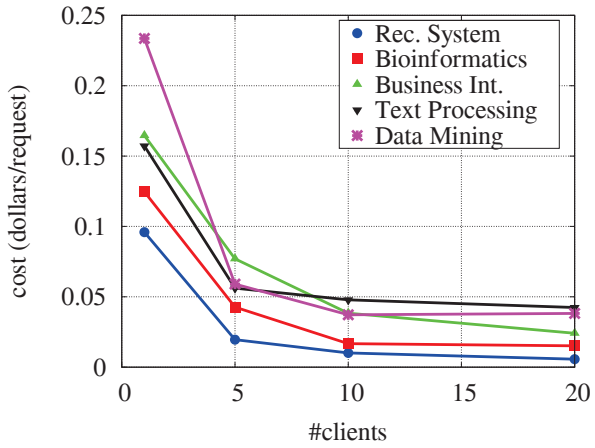
To illustrate how MRBS allows to evaluate the performance of MapReduce systems, we present here some experimental results. We perform these experiments in Amazon EC2 and vary the workload to see the impact on perfor-



(a) Throughput



(b) Client request response time



(c) Cost

Figure 4.6: Performance under different workloads

mance. Figure 4.6 shows the performance statistics obtained with a 20-node Hadoop cluster when the number of concurrent clients increases. All MRBS benchmarks are used in these experiments. Each benchmark uses its first dataload configuration as input data (see Table 4.2). The results presented in the following correspond to the average of three executions of 30 minutes run-time, after a 15 minutes of warm-up, with relative standard deviations of 0.2%-8%.

Figures 4.6(a) and 4.6(b) respectively present client request throughput and response time, for the different benchmarks. The throughput of the Recommendation System increases quasi-linearly with the number of concurrent clients. This is due to the fact that the MapReduce cluster is not overloaded and can thus cope with more concurrent clients. This is also confirmed by Figure 4.6(b) that shows that the response time of the Recommendation System does not increase with the number of concurrent clients. On the contrary, Text Processing cannot cope with increasing concurrent clients and response time increase linearly. Data Mining benchmark also shows a similar behaviour with increased workload.

The throughput of the Bioinformatics and Business Intelligence benchmarks increases linearly between 5 and 10 clients. The throughput with 20 clients continues to increase for Business Intelligence, whereas it does not increase appreciably with Bioinformatics, which reaches its maximum capacity. This is also reflected in the response times of the Bioinformatics and Business Intelligence benchmarks, which present a sharp increase between 10 and 20 clients. Thus, these experiments show that a MapReduce cluster is able to successfully host multi-user applications.

Interestingly, MRBS benchmarks show different throughput speedups, and this is explained as follows. Compared to the other benchmarks in MRBS, Recommendation System has the lowest average number of MapReduce tasks per job (see [Sangroya et al., 2012b] for more details). The lower the average number of MapReduce tasks per job in a benchmark is, the lower the number of task slots needed by that benchmark to run a request in the MapReduce cluster is, thus, the higher the number of available task slots for other concurrent client requests is, and the higher the benchmark throughput is. Consequently, thanks to concurrency in the MapReduce cluster, the average cost of a client request is reduced by a factor of up to 2-3, depending on the benchmark, cf. Figure 4.6(c).

Furthermore, MRBS also produces low-level MapReduce performance statistics, in terms of the number of MapReduce tasks per unit of time, and the size of data read or written in the distributed filesystem per unit of time, as respectively shown in Figures 4.7(a), 4.7(b) and 4.7(c). More specifically, the

Business Intelligence benchmark presents the highest amount of data read-/written. For reads, it is two orders of magnitude higher than Bioinformatics, and three orders higher than the Recommendation System. For writes, both Bioinformatics and Recommendation System write few data compared to the Business Intelligence benchmark, which is five orders of magnitude higher. This corroborates the results of client request response times, the Business Intelligence benchmark being the one with the highest response time.

4.5.3 Dependability Evaluation Under Faultloads

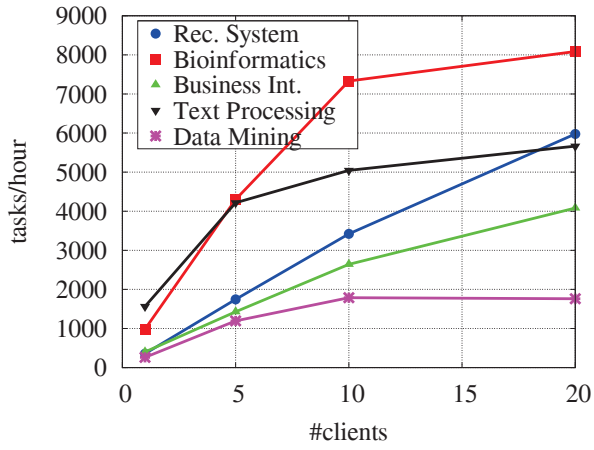
In this experiment, we illustrate the use of MRBS to evaluate the fault tolerance of Hadoop MapReduce. Here, a ten-node Hadoop cluster runs the Recommendation System and Business Intelligence benchmarks, with 20 concurrent clients, in the Grid'5000 cluster. The experiment is conducted during a run-time phase of 60 minutes, after a warm-up phase of 15 minutes. We consider a synthetic faultload that consists of software faults and hardware faults as follows: first, 100 map task software faults are injected 5 minutes after the beginning of the run-time phase, and then, 3 node crashes are injected 25 minutes later. Although the injected faultload is aggressive, the Hadoop cluster remains available 98% of the time for Recommendation System and 83% for Business Intelligence. The cluster is also able to successfully handle 97% of client requests for Recommendation System and 81% for Business Intelligence (see Table 4.4). This has an impact on the request cost which is 13% higher for Recommendation System and 33% for Business Intelligence than the cost obtained with the baseline (non-faulty) system.

Table 4.4: Reliability, availability, and cost.

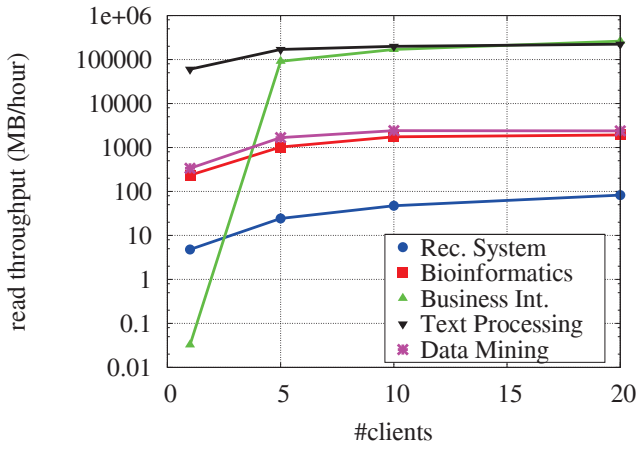
Benchmark	**Reliability**	**Availability**	**Cost (dollars/request)**
Rec. System	97%	98%	0.004 (+13%)
Business Int.	81%	83%	0.008 (+33%)

To better explain the behavior of the MapReduce cluster, we will analyze MapReduce statistics, as presented in Figures 4.8(a), 4.9(a) and 4.8(b), 4.9(b). Figures 4.8(a) and 4.9(a) presents successful MapReduce jobs and failed MapReduce jobs over time.

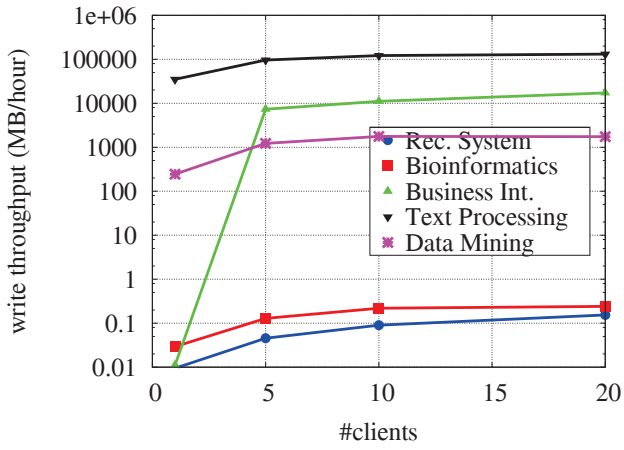
When software faults occur, few jobs actually fail. On the contrary, node crashes are more damaging and induce a higher number of job failures, with a drop of the throughput of successfull jobs from 16 jobs/minute before node failures to 5 jobs/minute after node failures.



(a) MapReduce task throughput



(b) DFS read throughput



(c) DFS write throughput

Figure 4.7: Low-level MapReduce statistics

Figures 4.8(b) and 4.9(b) show the number of successful MapReduce tasks and the number of failed tasks over time, differentiating between tasks that fail because they are unable to access data from the underlying filesystem (i.e. I/O failures in the figure), and tasks that fail because of runtime errors in all task retries[2] (i.e. task failures in the figure). We notice that software faults induce task failures that appear at the time the software faults occur, whereas node crashes induce I/O failures that last fifteen minutes after the occurrence of node faults. Actually, when some cluster nodes fail, Hadoop must reconstruct the state of the filesystem, by re-replicating the data blocks that were on the failed nodes from replicas in other nodes of the cluster[3]. This explains the delay during which I/O failures are observed.

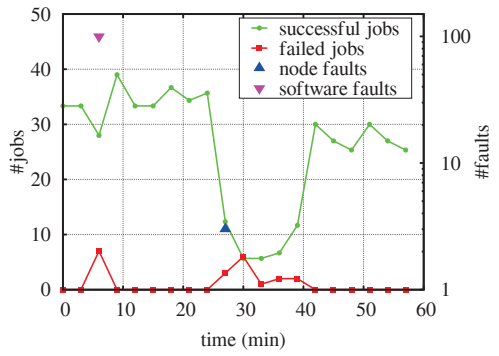
We now analyze the impact of these failures on the performance of the Hadoop MapReduce cluster. Figures 4.8(c) and 4.9(c)show the response time of successful client requests. With software faults, there is no noticeable impact on response times. Conversely, response time sharply increases when there are node faults, and while Hadoop is rebuilding missing data replicas. Similarly, Figures 4.8(d) and 4.9(d) depict the impact of failures on client request throughput. Interestingly, when the Hadoop cluster looses 3 nodes, it is able to fail-over, however, at the expense of a higher response time and a lower throughput.

4.6 Summary

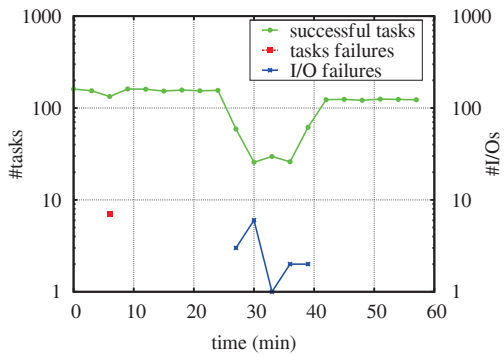
As cloud computing is evolving, many cloud services are provided. MapReduce is one of them, a popular Big Data cloud service, largely used by many companies. MapReduce provides a convenient means for distributed data processing and automatic parallel execution on clusters of machines. It has various applications and is used by several services featuring fault-tolerance and scalability. There has been a considerable interest in improving the dependability and performance of MapReduce. However, the ad-hoc and overly simplified setting used to evaluate most MapReduce fault-tolerance and performance improvement solutions poses significant challenges to the analysis and comparison of the effectiveness of these solutions.

In this chapter, we presented MRBS, a comprehensive benchmark suite for evaluating the dependability and performance of MapReduce systems. MRBS includes five benchmarks covering several application domains and a wide range of execution scenarios such as data-intensive vs. compute-intensive ap-

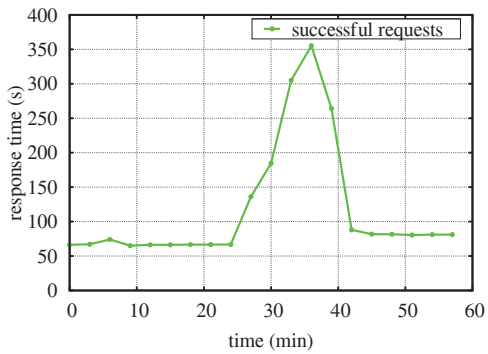
[2]By default, in Hadoop MapReduce, a task is executed at most four times before it fails.
[3]By default, data have three replicas in HDFS



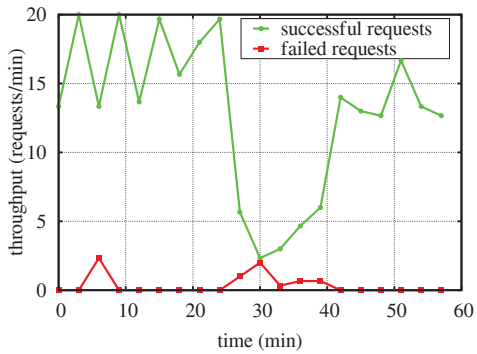
(a) Success vs. failed jobs



(b) Success vs. failed tasks

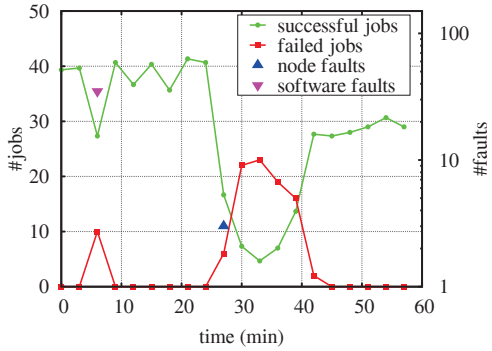


(c) Request response time

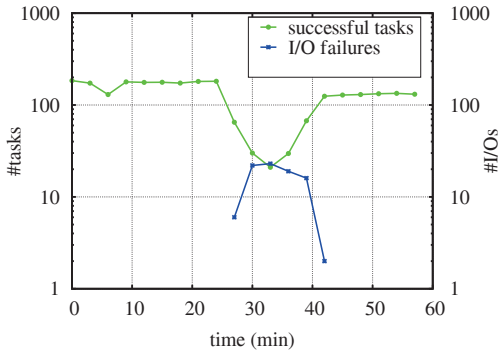


(d) Request throughput

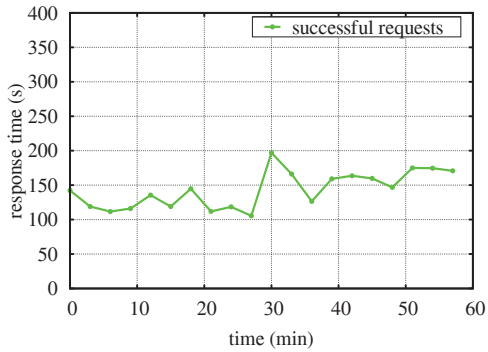
Figure 4.8: Dependability statistics for Recommendation System. Note the logarithmic scale of the right side y-axis.



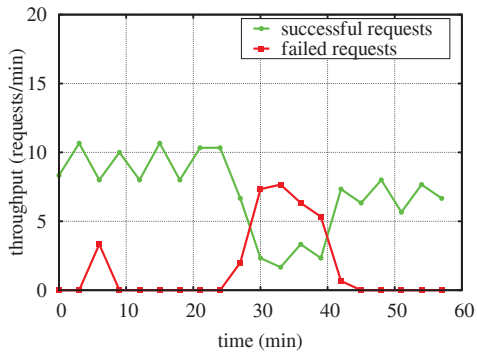
(a) Success vs. failed jobs



(b) Success vs. failed tasks



(c) Request response time



(d) Request throughput

Figure 4.9: Dependability statistics for Business Intelligence. Note the logarithmic scale of the right side y-axis.

plications, or batch applications vs. online interactive applications. MRBS allows to inject various workloads, dataloads and faultloads, and produces extensive reliability, availability and performance statistics. MRBS is available as a software prototype for Hadoop, a popular MapReduce framework available in public clouds. We also discuss the portability of MRBS on other MapReduce environments, and the automatic deployment of MRBS experiments on cloud infrastructures, which makes it easy to use. We implemented the MRBS benchmark suite for Hadoop MapReduce, and we illustrated its use to evaluate the performance and dependability of MapReduce systems.

This work opens interesting perspectives in terms of exploration of other fault models, and how to make MRBS open and extensible with other application domains and workloads. New heterogeneous workloads such as applications that possess low data read and high data write characteristics can also be integrated. We hope that such a benchmark suite will lead to less ad-hoc evaluations of MapReduce systems, and will help researchers and practitioners to better analyze and evaluate the fault-tolerance and performance of MapReduce.

Chapter 5

Use Cases

Contents

5.1 Introduction 89
5.2 Scalability With Regard To Cluster Size 89
5.3 Scalability With Regard To Data Size 91
5.4 How Many Faults Are Tolerated 91
5.5 Comparing Dependability of MapReduce Frameworks 93
5.6 Comparing Performance of MapReduce Frameworks 94
5.7 Summary 97

5.1 Introduction

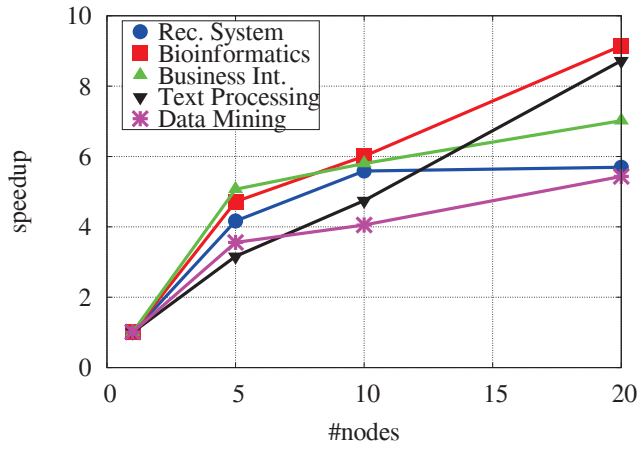
In Chapter 4, we introduced MRBS benchmark suite for performance and dependability benchmarking of MapReduce systems. MRBS has several possible uses, among which helping developers and testers to better analyze the fault-tolerance of MapReduce systems, or to better choose the configuration of the MapReduce cluster to provide service level guarantees. In this chapter, we illustrate the use of MRBS with six case studies which include, among others, the evaluation of the scalability of Hadoop clusters, and the comparison of performance and dependability levels of different Hadoop framework implementations. We use the same experimental setup as described in Section 4.5.1.

5.2 Scalability With Regard To Cluster Size

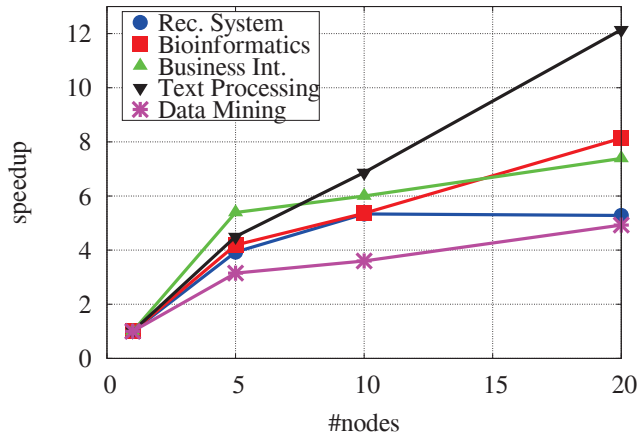
Here, we evaluate the scalability of Hadoop MapReduce with regard to the size of the Hadoop cluster. We conducted experiments with MRBS running on Hadoop clusters of different sizes: 5, 10, and 20 nodes, hosted in Amazon EC2. We compare the results of theses clusters with the results obtained when running MRBS on a one-node Hadoop cluster. We run all the benchmarks from MRBS with 10 concurrent clients accessing the cluster. For each experiment, average results of three runs are presented, with relative standard deviations below 4%. Figure 5.1 presents the performance results of the experiments. Figures 5.1(a) and 5.1(b) respectively show the response time speedup and throughput speedup, as functions of cluster size. The higher the response time speedup is, the better (i.e. lower) the response time is. Similarly, the higher the throughput speedup is, the better (i.e. higher) the throughput is.

Here, response time results show that, up to 5 nodes, the Hadoop cluster is able to scale linearly when it runs Bioinformatics or Business Intelligence. With higher cluster sizes, the speedup is sublinear. The Recommendation System benchmark achieve the maximum speedup with 10 nodes. This is not the case of other benchmarks. These differences in scalability capabilities are explained by the fact that other benchmarks have a much higher number of MapReduce tasks per client request than the Recommendation System (see [Sangroya et al., 2012b] for more details). Thus, the former benchmarks are able to exploit concurrency when having more nodes.

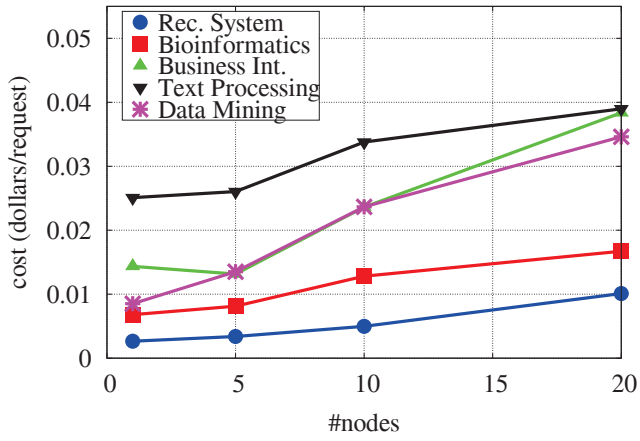
Figure 5.1(c) describes the average cost of a client request as a function of the number of nodes in Hadoop clusters. Obviously, the larger is the Hadoop cluster running on Amazon EC2, the higher is the cost of a client request. Surprisingly, the Business Intelligence benchmark shows that a five node Hadoop cluster costs less than one node. This is explained by the fact that with 5



(a) Response time speedup



(b) Throughput speedup



(c) Cost

Figure 5.1: Performance under different cluster scales

nodes, Business Intelligence achieves a superlinear speedup in throughput, and request cost is a function of throughput and Amazon EC2 hourly cost model.

5.3 Scalability With Regard To Data Size

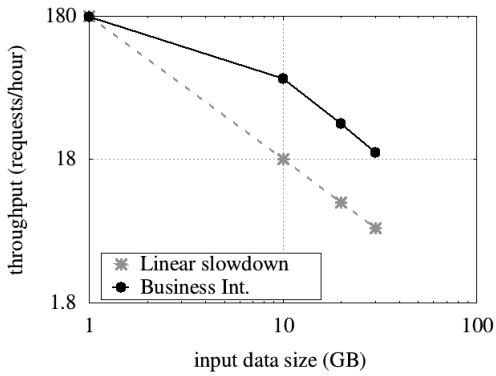
In the following, we investigate the scalability of Hadoop MapReduce with respect to the size of input data. We conducted experiments with the MRBS Business Intelligence benchmark, with different sizes of input data: 1 GB, 10 GB, 20 GB, and 30 GB. The experiments were conducted on a 20-node Hadoop cluster, hosted on Amazon EC2, with 10 concurrent clients. Each experiment is run three times to report average results, with relative standard deviations below 6%. Figure 5.2 presents performance results as functions of input data size. The measured performance of the benchmark is compared with the theoretical linear performance slowdown that we could expect when increasing the data size.

Figure 5.2(a) shows that, even though the performance decreases when the input data are larger, request throughput performs better than linear slowdown (note the logarithmic scale of the figure). Here, throughput is three times better than linear slowdown. This is due to the fact that accesses to the Hadoop filesystem do not increase linearly with the size of input data, as shown in Figure 5.2(b). Figure 5.2(b) describes the throughput of data read or written in the Hadoop filesystem, and Figure 5.2(c) describes MapReduce task throughput. These results show that with input data larger than 10 GB, the Hadoop cluster is overloaded, since it executes more MapReduce tasks while handling less client requests. This is also confirmed by other statistics reported by MRBS and showing an increase of MapReduce task retries, up to +25%, when input data are larger than 10 GB.

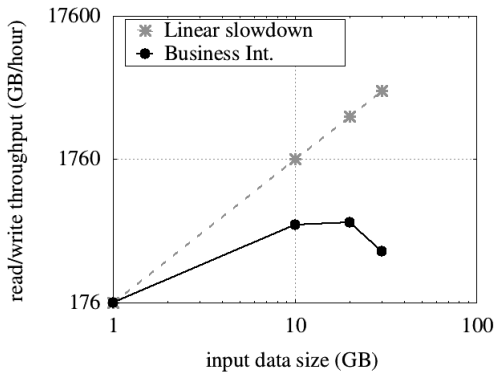
5.4 How Many Faults Are Tolerated

Here, we consider the case of a service provider that hosts MapReduce services on a ten-node cluster. One question that it has to answer would have the following form: *Up to how many node failures can the MapReduce cluster tolerate, while guaranteeing an availability of at least 85%?*

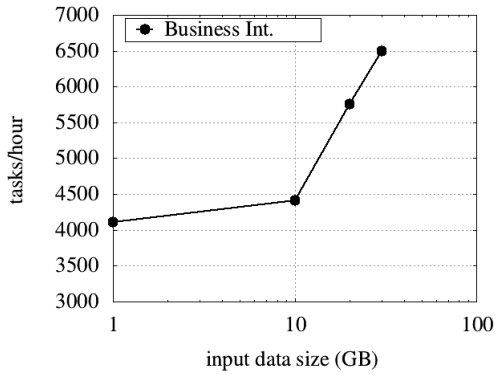
We conducted experiments with all benchmarks of MRBS, showing three different behaviour: the Business Intelligence service is data-intensive, the Recommendation System is compute-oriented, and the Bioinformatics service is in between (see Section 4.3). Figure 5.3 shows the measured availability,



(a) Request throughput



(b) Data read/write throughput



(c) MapReduce task throughput

Figure 5.2: Performance under different data scales

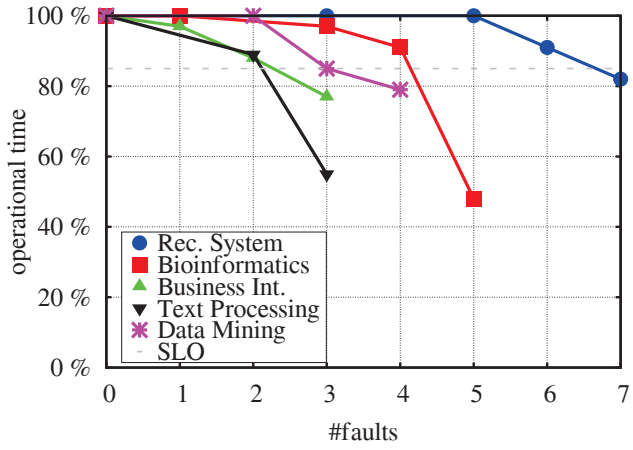


Figure 5.3: Availability under different node faultloads

with different faultloads. To guarantee the target availability objective of 85%, the MapReduce cluster hosting the data-intensive Business Intelligence service would not tolerate more than two node failures. In comparison, the less data-intensive Bioinformatics service would tolerate four node failures for the same availability objective, while the compute-oriented Recommendation System would be able to tolerate up to 6 node faults in a ten-node cluster. A similar behaviour is also shown with two micro-benchmarks i.e. Text Processing and Data Mining. There is relatively higher availability for Data Mining because it is more compute-intensive than Text Processing benchmark. In summary, Hadoop is able to transparently tolerate failures when there is one node crash. With more node failures, Hadoop MapReduce may handle failures with an acceptable availability level if the MapReduce service it hosts is more compute-intensive than data-intensive.

5.5 Comparing Dependability of MapReduce Frameworks

The goal of this experiment is to use MRBS to compare two different MapReduce framework implementations, with regard to their dependability and performance. Here, we considered two different versions of Hadoop, namely Hadoop v0.20.2 and Hadoop v1.0.0. Two sets of experiments were conducted on Grid'5000, hosting, on the one hand, a ten-node Hadoop v0.20.2 cluster, and on the other hand, a ten-node Hadoop v1.0.0 cluster. Each MapReduce cluster is used by 20 concurrent clients, running the MRBS Bioinformatics

benchmark. The default dataload was used (see Table 4.2). Each experiment consists of a 60 minute run-time phase, after a 15 minute warm-up phase. The same faultload was injected on each of the two MapReduce clusters; it consists of 100 MapReduce task software faults occurring after 5 minutes, and 3 node crashes occurring after 25 minutes.

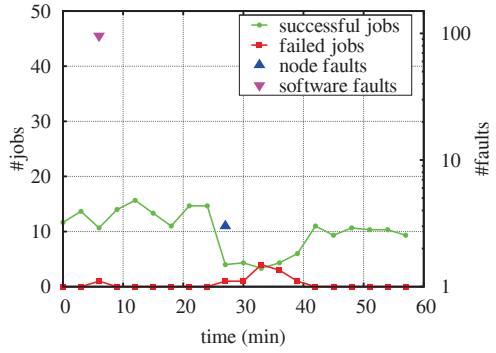
In figures 5.4 and 5.5, we show the comparison of dependability for Hadoop v0.20.2 and Hadoop v1.0.0. The two Hadoop frameworks provide similar behaviour in case of task software faults. However, Hadoop v1.0.0 provides higher fault tolerance than Hadoop v0.20.2 in case of node crashes. This is shown in Figures 5.4(a) and 5.5(a). This is explained by the fact that after node crashes, Hadoop v0.20.2 faces several I/O failures between 25 and 40 minutes (cf. Figure 5.4(b)), mainly due to the necessary time for MapReduce and the underlying filesystem to rereplicate lost data. This reconfiguration operation is apparently optimized in Hadoop v1.0.0, as shown in Figure 5.5(b). This has a direct impact on client request response times and throughput, as shown in Figures 5.4(c), 5.5(c), 5.4(d) and 5.5(d). Table 5.1 summarizes the results of dependability evaluation of two MapReduce frameworks. Hadoop v1.0.0 provides better dependability than Hadoop v0.20.2. This might be due to the additional security measures provided in the release of Hadoop v1.0.0 [Had, 2013].

Table 5.1: Dependability of two MapReduce frameworks.

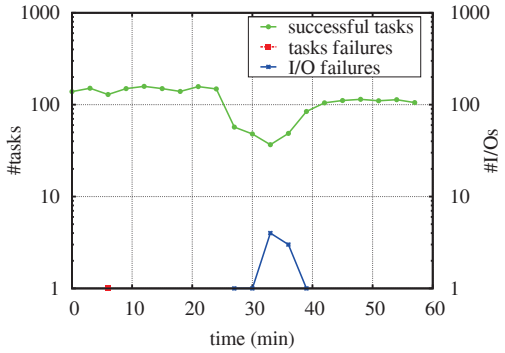
MapReduce framework	**Reliability**	**Availability**	**Cost (dollars/request)**
Hadoop v0.20.2	94%	96%	0.008 (+14%)
Hadoop v1.0.0	99%	99%	0.004 (+1%)

5.6 Comparing Performance of MapReduce Frameworks

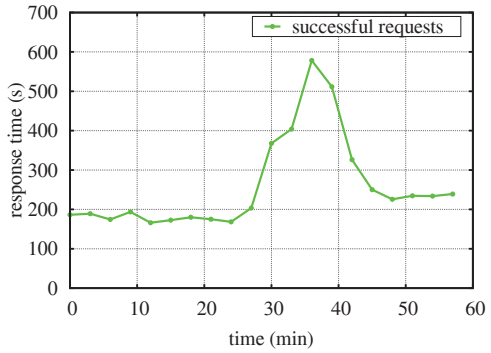
We also compared the two Hadoop v0.20.2 and Hadoop v1.0.0 MapReduce frameworks with regard to their performance. These experiments were conducted on Amazon EC2, hosting, on the one hand, a 10-node Hadoop v0.20.2 (blue bar), and on the other hand, a 10-node Hadoop v1.0.0 (red bar). With each setting, all benchmarks were used. Each MapReduce cluster is used by one client at a time with default dataloads (see Table 4.2). An experiment consists of a 15 minute run-time phase, after a 5 minute warm-up phase. Each



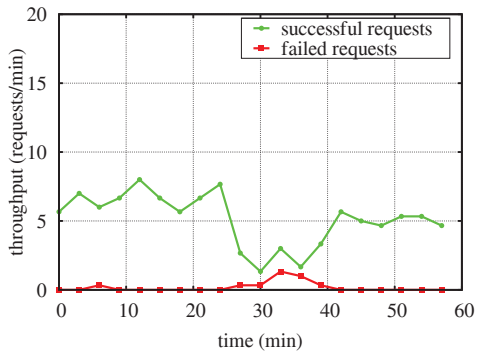
(a) Success. vs. failed jobs



(b) Success. vs. failed tasks

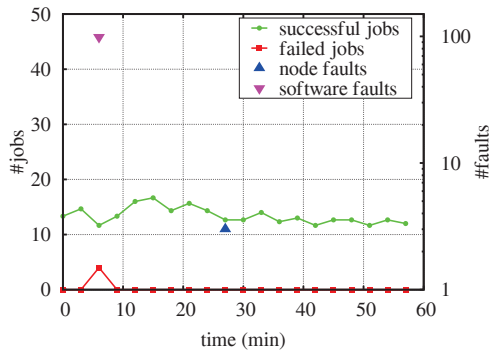


(c) Request response time

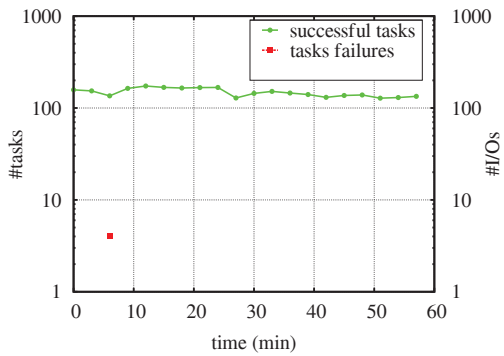


(d) Request throughput

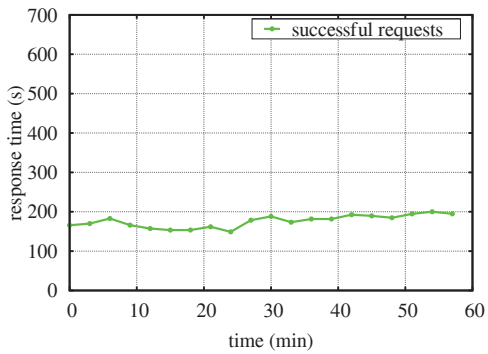
Figure 5.4: Dependability of two MapReduce frameworks – Bioinformatics, Hadoop v0.20.2



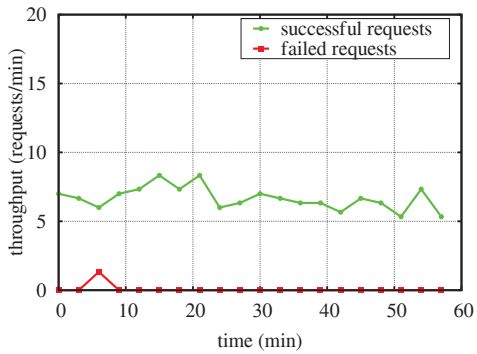
(a) Success. vs. failed jobs



(b) Success. vs. failed tasks



(c) Request response time



(d) Request throughput

Figure 5.5: Dependability of two MapReduce frameworks – Bioinformatics, Hadoop v1.0.0

experiment was run three times to report average results. No faultload was injected.

Figure 5.6 compares the client response times with the different MapReduce framework implementations. Surprisingly, Hadoop v1.0.0 provides lower performance (i.e. higher client response times) than Hadoop v0.20.2, whatever the benchmark is. Here, the average client response time with Hadoop v1.0.0 is higher than with Hadoop v0.20.2 by 34% for Recommendation System, 29% for Bioinformatics, 34% for Business Intelligence, 39% for Text Processing, and 27% for Data Mining benchmark.

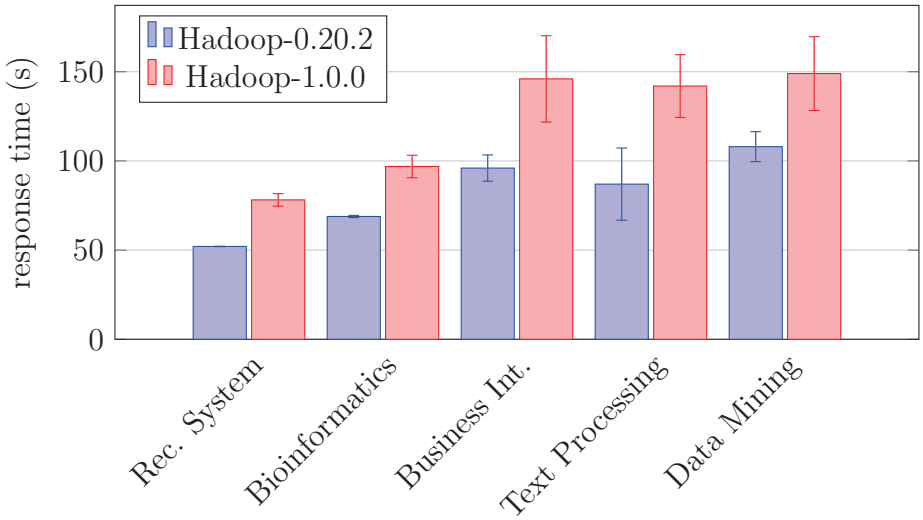


Figure 5.6: Performance of two MapReduce frameworks.

5.7 Summary

In this chapter, we illustrated the use of MRBS with case studies. MRBS can be used for different purposes such as evaluation of the scalability of Hadoop clusters, evaluation of the performance of Hadoop clusters, and comparison of performance and dependability levels of different Hadoop framework implementations. We conducted case studies from the perspective of a cloud service provider and cloud service user. This demonstrates the benefits of using a generic architecture to build performance and dependability benchmark for MapReduce cloud service.

Chapter 6

MemCB: Dependability & Performance Benchmarking for Memcached

Contents

6.1 Introduction **101**

6.2 Background **102**

6.2.1 Memcached 102

6.2.2 Fault Tolerance in Memcached 103

6.3 Overview of MemCB **104**

6.3.1 MemCB Workload 104

6.3.2 MemCB Dataload 105

6.3.3 MemCB Faultload 105

6.3.4 MemCB Fault Injection 105

6.3.5 Dependability and Performance Analysis in MemCB . 106

6.4 Experimental Evaluation **107**

6.4.1 Experimental Setup 107

6.4.2 Experimental Results 107

6.5 Summary **109**

6.1 Introduction

Memcached is widely used to provide in-memory caching solution for many popular web sites such as LiveJournal, Twitter, Flickr, Youtube and Facebook [Mem, 2013b]. There are some recent examples, where Memcached is being provided as a service in the cloud. Amazon provides ElastiCache, a web service that makes it easy to deploy, operate, and scale an in-memory cache in the cloud [Ela, 2013]. Similarly, there are others service providers such as Gear6 that provides high availability Memcached solutions [Gea, 2009, Mem, 2013e, Mem, 2013d]. Fault tolerance and high availability are among the prominent quality attributes advertised by these providers. For example, *Amazon ElastiCache* automatically detects and replaces failed nodes providing high reliability in the case of network overload and server crashes. Similarly, *Memcached Cloud* is a service for running Memcached in a reliable way [Mem, 2013d]. Such services guarantee that the data is constantly replicated, and if a node fails, a fail-over mechanism guarantees that data is served without any interruption.

There are some benchmarks to test the performance aspects of Memcached such as throughput or response time [Mem, 2013h, Ben, 2013, Mem, 2013i, Bru, 2009]. One limitation of these benchmarks is that they focus only on the performance aspects of Memcached. However, to test the fault tolerance capabilities, these tools do not provide any support. Indeed, while Memcached system was originally designed as a performance improvement solution, recent discussions in developer forums depicts the need to enhance the fault-tolerance features [Mem, 2013a]. For this reason, users need dependability and performance benchmarks to test the software applications developed over Memcached. Recent use of Memcached in high availability cloud services motivates us further to provide a benchmarking solution considering the dependability aspects of Memcached [Ela, 2013].

In this chapter, we introduce MemCB (MemCached Benchmarking), a dependability and performance benchmark for Memcached. We provide a simpler way for faultload injection in Memcached. This covers different fault types, such as node crashes and network faults, injected at different rates, that provide a means to analyze the fault-tolerance under different scenarios. MemCB includes a workload generation toolkit that emulates concurrent client requests in a Memcached cluster. We use MemCB to evaluate the performance and dependability levels of Memcached under different workloads, dataloads and faultloads.

The remainder of the chapter is organized as follows. Section 6.2 presents a background on Memcached and its fault tolerance capabilities. Section 6.3

provide an overview of MemCB, a description of performance and dependability analysis in MemCB. Section 6.4 describes the experimental evaluation and finally, Section 6.5 presents a summary of this chapter.

6.2 Background

6.2.1 Memcached

Memcached is a high-performance, distributed caching system which is commonly used to speed up dynamic web applications by lightening the database server load. This is done by speeding up the access to databases by storing the results of the database previous computations or any other data which is often accessed. Figure 6.1 shows a simple scenario, where first request goes to database server at the same time data object storing in Memcached server. After this, the second user request data comes from Memcached server. Memcached is easily deployable over existing applications. Memcached is simple and open-source. Therefore, it is widely used by high-traffic websites such as Youtube, Wikipedia, Facebook and others.

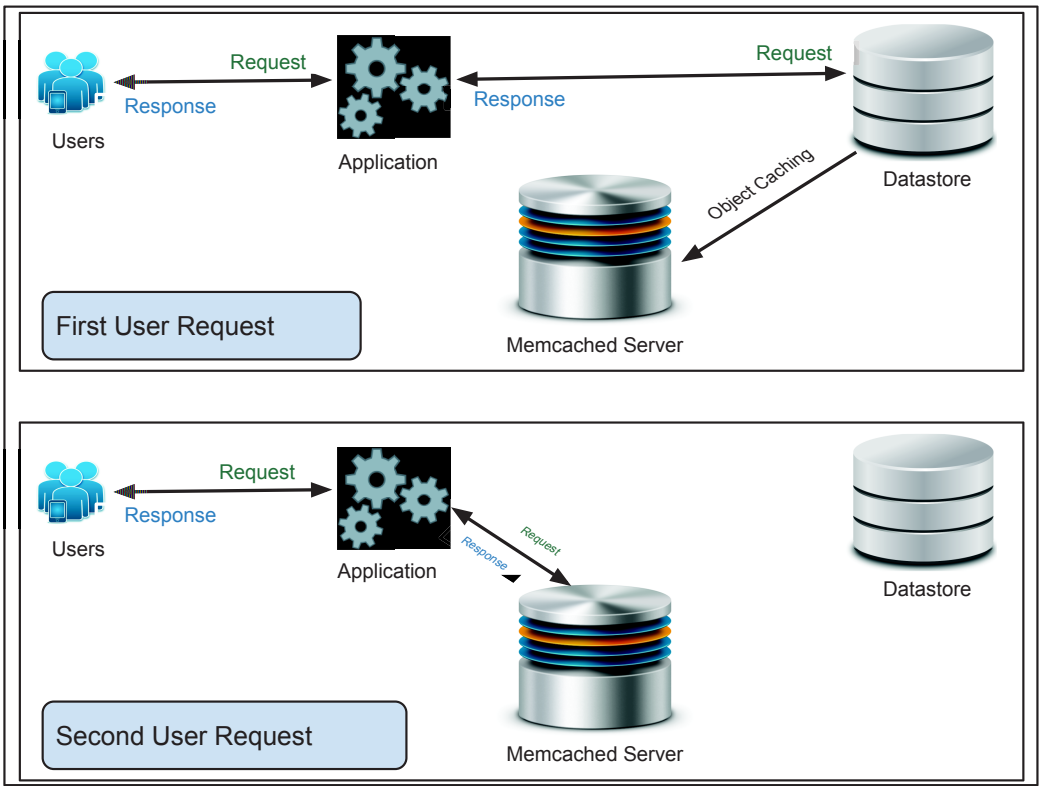


Figure 6.1: A simple Memcached workflow

Memcached works like a big hash table distributed across multiple machines. When the table is full, subsequent inserts cause older data to be purged in least recently used(LRU) order. Memcached's interface provides all the basic operations that hash tables provide, such as insertion, deletion, and lookup/retrieval; as well as more complex operations built upon them. The two basic operations are *GET*, to fetch the value of a given key, and *SET* to cache a value. Another common operation is to *DELETE* the key value pair as a way to invalidate the key if it was modified in persistent storage. In our work, we focused on read operations (GET requests) and write operations (SET requests) because they are the dominant operations in real world workloads such as at Facebook [Nishtala et al., 2013].

6.2.2 Fault Tolerance in Memcached

The primary design goal of Memcached is performance: to provide a high speed cache solution. However, we looked at some recent works that motivate to provide a high availability Memcached solution alongwith performance [Mem, 2007, Mem, 2012]. Memcached do not yet provide a rich fault tolerance model that handles a number of failures. We found that there are some faults to which Memcached provides a certain degree of fault tolerance. This is to guarantee that if any individual server node crash, there is an automatic mechanism to rebalance the cluster.

Memcached is able to tolerate failures of different types, as described in the following.

Hardware Crash. Version 2.0.0b2 of Memcached provides a mechanism for automatic fail-over. *OPT_AUTO_EJECT_HOSTS* is the tuning parameter that enables automatic fail-over in memcached systems. This supports transparent fail-over in case of a server node fails. Users need to set this option to *true* if they want transparent fail-over in case of failures [Mem, 2013c, Mem, 2013f].

Network Error. A server is marked as failed if it exceeds the timeout property. In the case, there is a high network load on a server and packets that are sent or received from a server node exceeds the timeout value, server is marked as dead. The parameter *OPT_RETRY_TIMEOUT* controls this error. Similar to the hardware crash, in this case, Memcached exclude this server from the list of servers as described before [Mem, 2013g].

6.3 Overview of MemCB

MemCB allows to inject various faultloads, workloads and dataloads in a Memcached system and to collect information helping testers understand the observed behavior of Memcached system. The overall architecture of MemCB is presented in Figure 6.2.

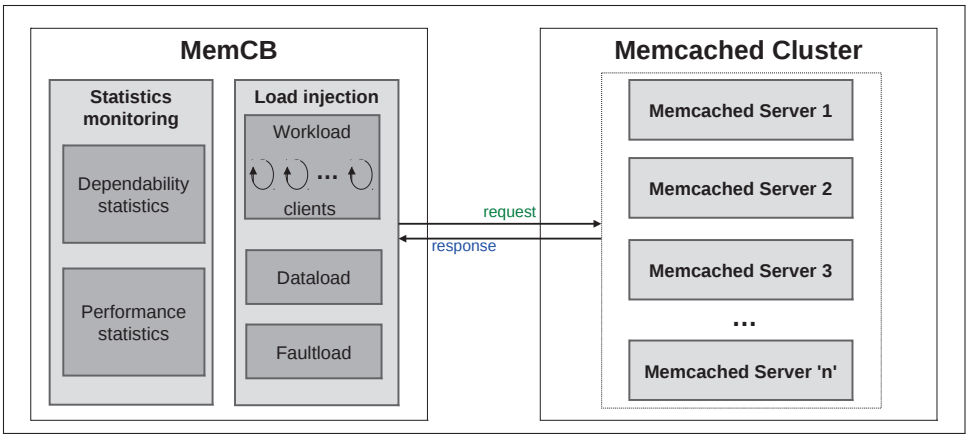


Figure 6.2: Overview of MemCB

MemCB comes with a benchmark, Brutis [Bru, 2009] that is a tool designed to test memcached instances by providing reproducible performance data for comparison purposes. Brutis is developed in PHP and is primarily a performance benchmark that helps by stressing a memcached cluster to test the workload and network characteristics. Brutis is able to generate client requests and produce some statistics related to performance. Some of the statistics generated by Brutis are: total numbers of operations, sets, set_fails, gets, hits, misses and the latency of the requests. Brutis use XML files to define the workload. It performs configurable sets of operations that store and retrieve data in the cache.

We adopt *Brutis* and perform two key operations: The *set* operation that store our generated data in Memcached and the *get* operation that retrieve the data given according to a key. These operations can be mixed to emulate more complicated scenarios.

6.3.1 MemCB Workload

As described in Chapter 3, the workload is characterized by the benchmark to execute, and the number of concurrent clients issuing requests on that benchmark application. MemCB workload consists of multiple concurrent clients issuing requests to the Memcached system. We use two kinds of client requests.

The *set* requests that write and store our generated data in Memcached; the *get* requests that retrieve the data from Memcached. In our experiments, 10% of the requests are *set* requests and 90% of the requests are *get* requests. This is explained in detail in Section 6.4 of this chapter.

6.3.2 MemCB Dataload

MemCB is built over Brutis, that provides a method to generate random data to be inserted into Memcached. Each key in the dataload is a concatenation of $prefix and $key_id. Users can choose to assign values to the key prefix and key ID.

6.3.3 MemCB Faultload

As discussed in 6.2.2, the faultload for Memcached consists of the following faults:

Node Crash. This fault is emulated as a hardware fault e.g. hard drive fault or a CPU crash. Interestingly, when this fault happens, Memcached looses any more any communication with the affected node and all the data is lost. Memcached client then copies all the data again on any of the other live nodes in the cluster.

Network Fault. The goal of this fault is to end/delay the communication over the network. This fault can also be visualized as a latency error. The node will suffer from an artificially emulated latency: meaning that the network is overloaded.

6.3.4 MemCB Fault Injection

The MemCB fault injector divides the input faultload into subsets of faultloads as follows: one global faultload that groups all faults that will occur in all nodes of the Memcached cluster (i.e. node crash, network fault).

The MemCB fault injector runs a daemon that is responsible of injecting the global faultload. Thus, for the i-th fault in the crash faultload, the dameon waits until $time_stamp_i$ is reached, then calls the fault injector of $fault_type_i$ (see below), on the Memcached cluster node corresponding to $fault_location_i$. This fault injector is called as many times as there are occurrences of the same fault at the same time. The fault injection dameon repeats these operations for the following crash faults, until the end of the faultload file is encountered or the end of the experiment is reached. Similar process is activated in the case of network faults.

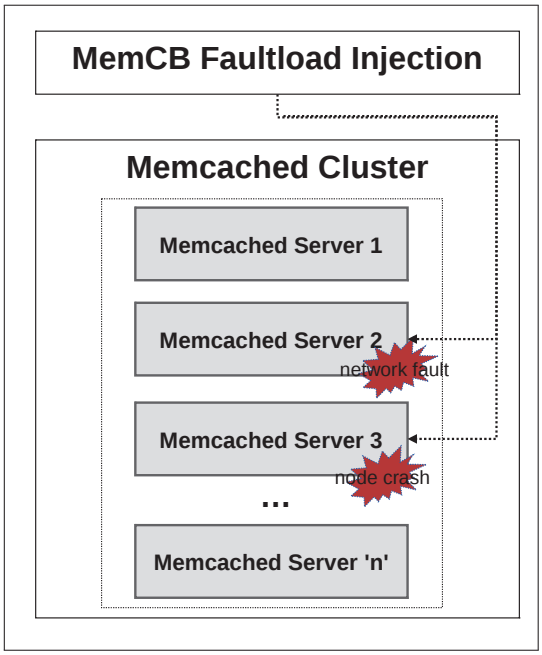


Figure 6.3: Architecture of MemCB faultload injector

The overall architecture of the faultload injection in MemCB is described in Figure 6.3.

Node Crash Injection. A node crash is simply implemented by shutting down a node. For efficiency purposes, we propose an implementation of Memcached node fault which kills all Memcached daemons running on that node. Then, the Brutis will not have any answer from Memcached and will consider this node as dead. The parameter, *OPT_AUTO_EJECT_HOSTS* is set to *true* that supports transparent fail-over.

Network Fault Injection. This type of fault is implemented through a tool named *tc* that injects latency on a network node [TC, 2013]. **TC** provides a way to manage the transmission of packets. In MemCB, to introduce a network fault, we create a delay for sending and receiving packets that further introduce an artificial latency.

6.3.5 Dependability and Performance Analysis in MemCB

MemCB produces runtime statistics related to dependability, such as reliability, and availability [Laprie, 1995]. Reliability is measured as the ratio of successful Memcached client requests to the total number of requests. This is measured as hit ratio, which is the ratio of number of hits to number of total requests. A hit/miss is defined as follows. When workload is searching for

some data in Memcached, and if it obtains the data, it will be considered as hit. Otherwise, if the data is no more there, the operation will be considered as a miss. Metrics for dependability is as follows:

cache hit rate: $\frac{number\ of\ hits}{total\ number\ of\ requests}$

In addition, MemCB produces performance statistics, such as client request response time, request throughput. Response time is the elapsed time from the moment the client submits a request until the response is received by the client. MemCB produces low level performance statistics as follows:

get request throughput: $\frac{numbers\ of\ get\ operations}{time}$

set request throughput: $\frac{numbers\ of\ set\ operations}{time}$

6.4 Experimental Evaluation

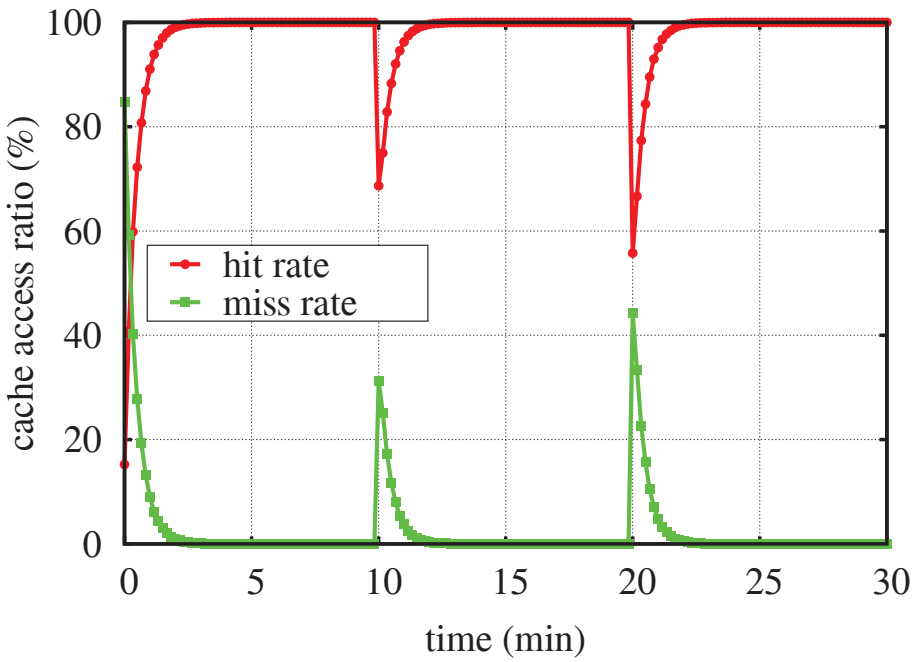
In the following, we illustrate the experiments with MemCB to evaluate the performance and dependability of Memcached.

6.4.1 Experimental Setup

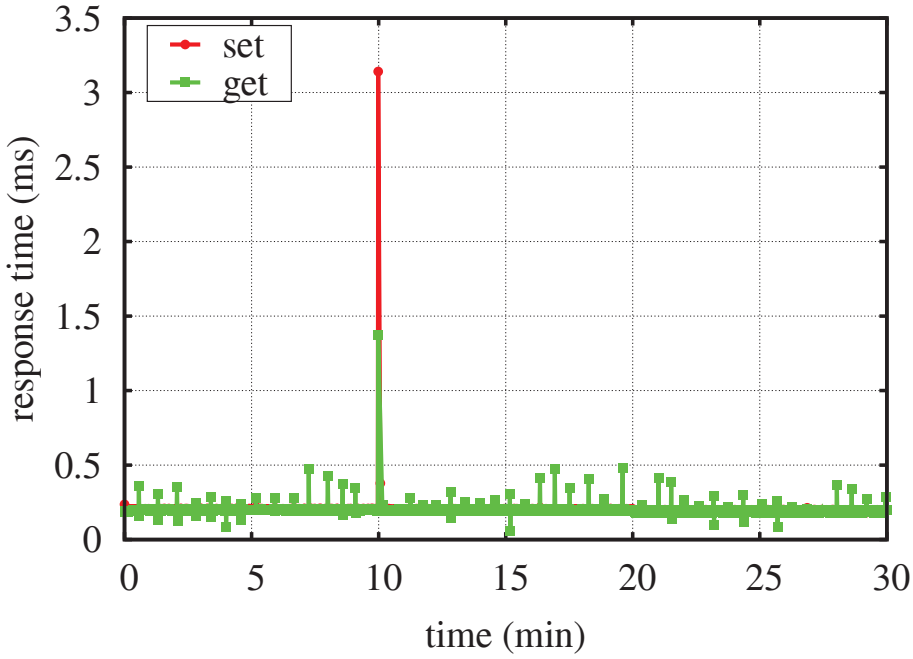
The experiments presented in this section were conducted in a cluster running on Grid'5000 [Bolze et al., 2006]. Each cluster consists of one node hosting MemCB and emulating workload, and a set of nodes hosting the Memcached cluster. The hardware configuration consists of 4-core 2-CPU, 2.5 GHz Intel Xeon E5420 QC CPU, 8 GB memory, 160 GB SATA storage (per node) and 1 GB Ethernet network. The software environment consists of Libmemcached-1.0.16, Memcached-1.4.15, and PHP version 2.1.0. Each node run Debian Linux 6 with kernel v2.6.32.

6.4.2 Experimental Results

Now, we present the results of using MemCB to evaluate the dependability and performance of Memcached. Following experiments were conducted on a four-nodes Memcached cluster. One node is used to host MemCB which emulates workload that consists of 10 clients per node sending client requests in a random manner. 10% of the requests are *set* requests and 90% of the requests are *get* requests. The experiment is conducted during a run-time phase of 30 minutes, including a warm-up phase. We consider a synthetic faultload that consists of network faults and hardware faults as follows: first, network faults are injected 10 minutes after the beginning of the run-time



(a) Cache access ratio



(b) Response time

Figure 6.4: Dependability and performance of Memcached under network faults and node crashes

phase, and then node crashes are injected 10 minutes later. To better explain the behavior of the Memcached cluster, we will analyze statistics, as presented in Figures 6.4(a) and 6.4(b).

Figure 6.4(a) presents successful Memcached hits and misses over time. When network faults occur, the server node on which we injected the fault, fails to send and receive the packets within the timeout limit (a value configured in the Memcached). As a result of this, after certain number of server retries, the node which was taking long time to respond, is marked as failed. Memcached removes this node from the list of available nodes. Thereafter, Memcached client copies the data (keys) on the other live nodes in the cluster.

We can observe from Figure 6.4(b) that at the time of network fault injection (at 10 minutes), there is an impact on the response time of client requests. High response time during this period is due to the injected latency, because of which some client requests are successful but they took long time to respond. On the contrary, with node crash fault injection (at 20 minutes) we observe a slightly similar behaviour of cache access ratio. This is because, similar to the network fault where the node was marked as failed due to timeout, after a node crash, Memcached removes the faulty nodes from the list of available nodes. We also observe that number of hits decrease and number of misses increase when we injected the node crash fault (compared to the period when we inject network fault). This is because of higher load on the servers after one node was removed after network fault. There was one less node after network fault and load per server was higher.

6.5 Summary

To evaluate the dependability and performance of Memcached systems, we developed a software prototype, MemCB that covers realistic workloads, dataloads and faultloads and allows their injection in a running Memcached cluster. In this chapter, we have presented the details of this prototype which is based on the principals discussed in Chapter 3. MemCB allows the empirical evaluation of Memcached systems to quantify their dependability levels in terms of reliability, and their performance levels in terms of response times and throughput. It allows to inject a rich faultload in a running Memcached cluster.

Experimental results confirm that MemCB is able to successfully evaluate the performance and dependability aspects of a Memcached system. The prototype is intended to be used by the users of Memcached service in the cloud. One of the future work is to add more workloads and dataloads so that the benchmark is richer. Important perspective of this chapter is the addition

of other fault types such as software faults. More experiments under variety of scenarios are also needed to evaluate and improve the proposed software prototype.

Chapter 7

Conclusions and Perspectives

Contents

7.1 Conclusions 113

7.2 Perspectives 115

7.1 Conclusions

In this thesis, we focused on the performance and dependability benchmarking approach for cloud computing services. We argue that traditional solutions to achieve and evaluate high dependability may not be appropriate for modern applications in cloud computing. The ideas presented in this thesis benefit both cloud service providers and users. Cloud service providers can easily integrate such benchmarks for dependability and performance analysis in their services. Cloud service users on the other hand, can get benefits by using the benchmark for dependability and performance analysis. Till now, ad-hoc and overly simplified settings are being used to evaluate most cloud service fault-tolerance and performance improvement solutions. This poses significant challenges to the analysis and comparison of the effectiveness of these solutions. A benchmark could be a very useful tool to systematically evaluate the performance and dependability for cloud services.

We studied various research, scientific and technical issues in designing a dependability and performance benchmark for cloud services in this thesis. To support our idea, we designed, implemented, and evaluated benchmarks for services belonging to PaaS model of cloud computing. We proposed a generic architecture which is further used to build two software prototypes: MRBS and MemCB. These prototypes were used to benchmark two popular cloud services: MapReduce and Memcached. The architecture description included the details of workload, dataload and faultload components, for benchmarking dependability and performance of a cloud computing service.

Having available a generic software architecture greatly helps the designers of dependability benchmarking solutions in reducing the efforts to design and develop new benchmarks for cloud services. This architecture helps to reduce the efforts needed to build a new dependability benchmark from scratch. The proposed architecture can be used by the designers of dependability benchmark solutions for cloud computing services. The software prototypes MRBS and MemCB can be directly used to analyze dependability and performance of MapReduce systems and Memcached service respectively.

MRBS benchmarking suite allows the empirical evaluation of MapReduce systems to quantify their dependability and performance levels. MRBS covers five application domains and a wide range of execution scenarios, workloads and dataloads. It also allows to characterize a faultload, generate it, and inject it in a running MapReduce cluster. We illustrated the use of MRBS with six case studies which include, among others, the evaluation of the scalability of Hadoop clusters, and the comparison of performance and dependability levels of different Hadoop framework implementations. MRBS is available as

a software prototype for Hadoop, a popular MapReduce framework available in public clouds. MRBS could be used for academic; research and industrial purposes such as for benchmarking an in-house developed map-reduce framework.

Similarly, MemCB allows the empirical evaluation of dependability and performance of Memcached systems. The workloads, dataloads and faultloads used in MemCB are realistic and represent real world scenarios. Both MRBS and MemCB prototypes follows the design rules of the generic architecture introduced in Chapter 3. MemCB allows to inject a rich faultload in a running Memcached cluster. Experimental results depicted that MemCB is able to successfully evaluate the performance and dependability aspects of a Memcached system. The addition of other fault types and experimental scenarios of MemCB are considered as future work.

The distribution of the prototypes developed as part of this thesis is free, so that the scientific and industrial communities can reuse the work. MRBS software prototype can be downloaded from: `http://sardes.inrialpes.fr/research/mrbs`. In Table 1.2, we demonstrated the statistics from the MRBS website. Between June 5, 2012 and January 1, 2014, MRBS webpage is visited 980 times (including 550 unique visitors). The results of this work have been featured in publications and scientific events as following.

- A. Sangroya, D. Serrano and S. Bouchenak. **Experience with Benchmarking Dependability and Performance of MapReduce Systems**. *IEEE Transactions on Dependable and Secure Computing (TDSC), under submission*

- A. Sangroya, D. Serrano, S. Bouchenak. **Benchmarking Dependability of MapReduce Systems**. *31st IEEE International Symposium on Reliable Distributed Systems (SRDS). Irvine, California, Oct 2012*

- A. Sangroya, D. Serrano and S. Bouchenak. **MRBS: Towards Dependability Benchmarking for Hadoop MapReduce**. *Workshop on Big Data Management in Clouds (BDMC) in conjunction with Euro-Par. Rhodes Island, Greece, Aug. 2012*

- A. Sangroya, D. Serrano, S. Bouchenak. **Towards a Dependability Benchmark Suite for MapReduce**. *Poster at EuroSys 2012, Bern, Switzerland. Apr 2012*

- L. Lemke, A. Sangroya, D. Serrano and S. Bouchenak. **Evaluer la tolérance aux fautes de systèmes MapReduce**. *Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS). Grenoble, France, Jan 2013*

7.2 Perspectives

This work opens a number of scientific and technical perspectives. First, the dependability and performance benchmark can be enriched with other metrics such as data security etc. This work opens interesting perspectives in terms of exploration of other fault models, and how to make prototypes such as MRBS open and extensible with other application domains and workloads. In this thesis, we considered performance and dependability metrics such as response time, throughput, reliability, availability respectively in addition to financial cost metrics. We believe that the addition of new metrics would greatly benefit the benchmark users who want to evaluate other attributes of cloud services.

The framework and benchmark prototypes developed as part of this thesis can also be used as a supporting tool for quality of service (QoS) management for cloud services. Service providers who want to guarantee a certain performance and dependability levels, as part of a Service Level Agreement can make use of our prototypes to plan the capacity of cloud services. One such example is from *Serrano et al.* [Serrano et al., 2013], where the authors consider the online control of cloud services to provide performance, dependability and cost guarantees.

The framework proposed in this work could be proposed in other use cases. Hadoop that supports MapReduce comes with a number of configuration parameters. It is often difficult to choose an optimal setting for these parameters. MRBS could be used to run Hadoop with different configuration values, look at the impact on quality attributes such as performance and dependability, and find the best values for these parameters. MRBS can be useful tool in such situations, and provides an easy way to run benchmarks on Hadoop and study the impact on performance and dependability. Another interesting application is to use MRBS to compare different scheduling algorithms. In general, the benchmark prototypes can be used for different purposes by the cloud service providers and users. The framework proposed in this work could also be considered for other cloud services such as IaaS and SaaS.

The benchmarks proposed in this thesis to evaluate performance and dependability can be supported by a verification and validation module. In our experimental evaluation, we have made use of statistical measures such as average, mean, etc., to validate our results. As part of a future work, we believe that more formal approaches of evaluation can be used to guarantee the correctness of the results. A technical perspective of this work could be to provide benchmark suites that focus on other important areas of cloud services such as database, operating system, virtual machines etc.

Acknowledgments

This work was partly supported by the Agence Nationale de la Recherche, the French National Agency for Research, under the MyCloud ANR project (http://mycloud.inrialpes.fr/). Part of the experiments were conducted on the Grid'5000 experimental testbed, developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (http://www.grid5000.fr).

Appendix A

Annexes

Contents

A.1 Properties file of MRBS 119

A.1 Properties file of MRBS

Following is a small example of MRBS properties file. This file contains the value of various configuration parameters of the benchmark. The properties file is designed for usability. For ease of use, most of the configuration parameters have default values. Moreover, to configure most of the parameters, users have to just comment/uncomment the associated line. For few parameters, specific values must be provided such as for run time and warm up time of the benchmark. This information is also provided on the website http://sardes.inrialpes.fr/research/mrbs.

```
# Choose (i.e. uncomment) one among the following.

##################################################
# Benchmark length and concurrency level
##################################################

# Warm-up phase length (in seconds): Time for the warm up
warm.up.time = 600

# Run-time phase length (in seconds): Time for the run
run.time = 1200

# Number of con. clients
number.concurrent.clients = 100

# Workload execution mode (Interactive mode: concurrent
   clients share cluster resources)
#execution = batch
execution = interactive

##################################################
# Configuration of the cloud
##################################################

# Cloud type: Name of the cloud
cloud.name = ec2
#cloud.name = grid5000
#cloud.name = local

# Cloud Instance Type (For Amazon EC2)
instance.type = t1.micro
```

```
#instance.type = m1.large

# Financial cost (in US dollars per machine*hour)
hourly.cost = 0.32

##################################################
# Underlying software paths
##################################################

# MRBS home
mrb.path = /home/sangroya/svn_mrbs/Software/mrbs

# Hadoop MapReduce framework home
mapreduce.framework.path = /home/sangroya/sango/hadoop

##################################################
# Benchmark workload
##################################################

# Recommendation System Benchmark
workload.mix = recommendation_system

# Decision Support Benchmark
#workload.mix  = decision_support

# DNA Sequencing Benchmark
#workload.mix = dna_sequencing

##################################################
# Benchmark faultload
##################################################

# Faultload generation (synthetic, random OR trace-based)
faultload.generation = synthetic
#faultload.generation = random
#faultload.generation = trace-based
```

Bibliography

[DBe, 2004] (2004). Dependability Benchmarking Project. http://webhost.laas.fr/TSF/DBench/.

[Med, 2006] (2006). MediaBench II Benchmark. http://euler.slu.edu/~fritts/mediabench/.

[Mem, 2007] (2007). Memcached Fault Tolerance. http://grokbase.com/t/danga/memcached/079vsmj7mt/fault-tolerance.

[Bru, 2009] (2009). Brutis Memcache benchmarking tool written in PHP. http://code.google.com/p/brutis/wiki/Readme.

[Gea, 2009] (2009). MySQL Scaling with Memcached. http://www.slideshare.net/gear6memcached/gear6-webinar-mysql-scaling-with-memcached.

[20N, 2011] (2011). 20 Newsgroups. http://people.csail.mit.edu/jrennie/20Newsgroups/.

[Ama, 2011a] (2011a). Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/.

[Ama, 2011b] (2011b). Amazon Elastic MapReduce. http://aws.amazon.com/elasticmapreduce/.

[Had, 2011a] (2011a). Apache Hadoop. http://hadoop.apache.org/.

[App, 2011] (2011). Appistry Inc. Cloud MapReduce. http://www.appistry.com/go/cmr.

[Fai, 2011] (2011). Fair Scheduler. https://hadoop.apache.org/docs/r1.2.1/fair_scheduler.html.

[Had, 2011b] (2011b). Fault injection framework. http://hadoop.apache.org/hdfs/docs/r0.21.0/faultinject_framework.

[San, 2011] (2011). Genomic research centre. http://www.sanger.ac.uk/.

[Goo, 2011] (2011). Google App Engine. http://code.google.com/intl/en/appengine/.

[Gre, 2011] (2011). Greenplum. http://www.greenplum.com/.

[Gri, 2011] (2011). Gridmix3 Emulating Production Workload for Apache Hadoop. http://developer.yahoo.com/blogs/hadoop/posts/2010/04/gridmix3_emulating_production/.

[HPC, 2011] (2011). HPC Challenge Benchmark. http://icl.cs.utk.edu/hpcc/.

[Mov, 2011] (2011). MovieLens web site. http://movielens.umn.edu/.

[Ope, 2011a] (2011a). Open Cirrus: The HP/Intel/Yahoo! Open Cloud Computing Research Testbed. https://opencirrus.org/.

[Ope, 2011b] (2011b). OpenStack. http://www.cloud.com.

[Pig, 2011] (2011). PigMix. http://cwiki.apache.org/confluence/display/PIG/PigMix.

[Inf, 2011] (2011). The 10 worst cloud outages (and what we can learn from them). http://www.infoworld.com/d/cloud-computing/the-10-worst-cloud-outages-and-what-we-can-learn-them-902.

[TPC, 2011a] (2011a). TPC Benchmark H - Standard Specification. http://www.tpc.org/tpch/.

[TPC, 2011b] (2011b). TPC-C: an On-Line Transaction Processing Benchmark. http://www.tpc.org/tpcc/.

[Mem, 2012] (2012). Memcached for PHP and failover. http://stackoverflow.com/questions/12372031/memcached-for-php-and-failover.

[CRN, 2012] (2012). The 10 Biggest Cloud Outages Of 2012. http://www.crn.com/slide-shows/cloud/240144284/the-10-biggest-cloud-outages-of-2012.htm.

[Wik, 2012] (2012). Wikipedia Dump. http://meta.wikimedia.org/wiki/Data_dumps.

[Ela, 2013] (2013). Amazon ElastiCache. http://aws.amazon.com/elasticache.

[Ama, 2013] (2013). Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/.

[Hiv, 2013] (2013). Apache Hive. http://hive.apache.org/.

[Mah, 2013] (2013). Apache Mahout. http://mahout.apache.org.

[Ben, 2013] (2013). Benchmarking Memcached and Redis clients. http://alekseykorzun.com/post/53283070010/benchmarking-memcached-and-redis-clients.

[EEM, 2013] (2013). Embedded Microprocessor Benchmark Consortium . http://www.eembc.org/.

[Had, 2013] (2013). Hadoop 1.0.0 Release Notes. http://hadoop.apache.org/docs/r1.0.0/releasenotes.html.

[Mem, 2013a] (2013a). Hosted fault-tolerant memcache solution in amazon ec2 cloud? http://serverfault.com/questions/333572/hosted-fault-tolerant-replicated-memcache-solution-in-amazon-ec2-cloud.

[Sma, 2013] (2013). IBM SmartCloud. http://www-935.ibm.com/services/in/en/cloud-enterprise/.

[Mem, 2013b] (2013b). Memcached. http://memcached.org.

[Mem, 2013c] (2013c). Memcached 2.0.0b2. http://pecl.php.net/package/memcached/2.0.0b2.

[Mem, 2013d] (2013d). Memcached Cloud. https://devcenter.heroku.com/articles/memcachedcloud.

[Mem, 2013e] (2013e). Memcached Comparison. http://garantiadata.com/memcached/memcached-comparison.

[Mem, 2013f] (2013f). Memcached Construct. http://php.net/manual/en/memcached.construct.php.

[Mem, 2013g] (2013g). Memcached getResultCode. http://us3.php.net/manual/en/memcached.getresultcode.php.

[Mem, 2013h] (2013h). Memslap - Load testing and benchmarking a server. http://docs.libmemcached.org/bin/memslap.html.

[Mem, 2013i] (2013i). Memtier benchmark - A High-Throughput Benchmarking Tool for Redis and Memcached. http://garantiadata.com/blog/memtier_benchmark-a-high-throughput-benchmarking-tool-for-redis-memcached.

[Nim, 2013] (2013). Nimbus is cloud computing for science. http://www.nimbusproject.org/.

[Rac, 2013] (2013). Rackspace Public Cloud. http://www.rackspace.com/cloud/.

[SPE, 2013] (2013). Standard Performance Evaluation Corporation. http://www.spec.org/.

[Net, 2013] (2013). The worst cloud outages of 2013 (so far). http://www.networkworld.com/slideshow/108568/the-worst-cloud-outages-of-2013-so-far.html.

[TC, 2013] (2013). Traffic Control HOWTO. http://www.tldp.org/HOWTO/html_single/Traffic-Control-HOWTO/.

[VMm, 2013] (2013). VMware VMmark 2.x. http://www.vmware.com/products/vmmark/overview.html.

[Win, 2013] (2013). Windows Azure. http://www.windowsazure.com/en-us/.

[Abouzeid et al., 2009] Abouzeid, A., Bajda-Pawlikowski, K., Abadi, D., Rasin, A., and Silberschatz, A. (2009). HadoopDB: An Architectural Hybrid of MapReduce and DBMS Technologies for Analytical Workloads. In *International Conference on Very Large Data Bases (VLDB)*.

[Agarwal and Prasad, 2012] Agarwal, D. and Prasad, S. (2012). AzureBench: Benchmarking the Storage Services of the Azure Cloud Platform. In *IEEE 26th International Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012*, pages 1048–1057.

[Alexandrov et al., 2012] Alexandrov, A., Folkerts, E., Sachs, K., Iosup, A., Markl, V., and Tosun, C. (2012). Benchmarking in the Cloud: What it Should, Can, and Cannot Be. In *TPC Technology Conference on Performance Evaluation and Benchmarking*, TPCTC'12.

[Ananthanarayanan et al., 2011] Ananthanarayanan, G., Agarwal, S., Kandula, S., Greenberg, A., Stoica, I., Harlan, D., and Harris, E. (2011). Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters. In *EuroSys*.

[Armbrust et al., 2010] Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A View of Cloud Computing. *ACM Communications*, 53(4):50–58.

[Barbosa et al., 2011] Barbosa, R., Karlsson, J., Yu, Q., and Mao, X. (2011). Toward Dependability Benchmarking of Partitioning Operating Systems. In *IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN), 2011*, pages 422 –429.

[Bessani et al., 2010] Bessani, A. N., Cogo, V. V., Correia, M., Costa, P., Pasin, M., Silva, F., Arantes, L., Marin, O., Sens, P., and Sopena, J. (2010). Making Hadoop MapReduce Byzantine Fault-Tolerant. In *IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), Fast abstract*.

[Bolze et al., 2006] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., and Touche, I. (2006). Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *International Journal of High Performance Computing Applications*, 20(4):481–494.

[Bond et al., 2013] Bond, A., Kopczynski, G., and Taheri, H. (2013). Two Firsts for the TPC: A Benchmark to Characterize Databases Virtualized in the Cloud, and a Publicly-Available, Complete End-to-End Reference Kit. In *Selected Topics in Performance Evaluation and Benchmarking*, volume 7755 of *Lecture Notes in Computer Science*, pages 34–50.

[Bressan et al., 2003] Bressan, S., Lee, M., Li, Y., Lacroix, Z., and Nambiar, U. (2003). The XOO7 Benchmark. In *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web*, volume 2590 of *Lecture Notes in Computer Science*, pages 146–147. Springer Berlin Heidelberg.

[Brown and Patterson, 2000] Brown, A. and Patterson, D. A. (2000). Towards Availability Benchmarks: A Case Study of Software RAID Systems. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, ATEC '00, pages 22–22.

[Böhme and Rahm, 2001] Böhme, T. and Rahm, E. (2001). XMach-1: A Benchmark for XML Data Management. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, Informatik aktuell, pages 264–273. Springer Berlin Heidelberg.

[Carey et al., 1993] Carey, M. J., Dewitt, D. J., and Naughton, J. F. (1993). The 007 Benchmark. In *Proceedings of the 1993 ACM SIGMOD International conference on Management of data*, SIGMOD '93, pages 12–21, New York, NY, USA. ACM.

[Cattell and Skeen, 1992] Cattell, R. G. G. and Skeen, J. (1992). Object Operations Benchmark. *ACM Transactions on Database Systems (TODS)*, 17(1):1–31.

[Cerbelaud et al., 2009] Cerbelaud, D., Garg, S., and Huylebroeck, J. (2009). Opening the Clouds: Qualitative Overview of the State-of-the-art Open Source VM-based Cloud Management Platforms. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, pages 22:1–22:8, New York, NY, USA. Springer-Verlag New York, Inc.

[Che et al., 2009] Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IEEE International Symposium on Workload Characterization (IISWC 2009)*, pages 44–54.

[Chen and Schlosser, 2008] Chen, S. and Schlosser, S. W. (2008). MapReduce Meets Wider Varieties of Applications. Technical Report IRP-TR-08-05, Intel.

[Chen et al., 2011] Chen, Y., Ganapathi, A., Griffith, R., and Katz, R. (2011). The Case for Evaluating MapReduce Performance Using Workload Suites. In *IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*.

[Chung and Hollingsworth, 2004] Chung, I.-H. and Hollingsworth, J. (2004). Automated cluster-based Web service performance tuning. In *Proceedings of 13th IEEE International Symposium on High performance Distributed Computing, 2004*, pages 36–44.

[Condie et al., 2010] Condie, T., Conway, N., Alvaro, P., Hellerstein, J., Elmeleegy, K., and Sears, R. (2010). MapReduce Online. In *USENIX Symp. on Networked Systems Design and Implementation (NSDI)*.

[Cooper et al., 2010] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., and Sears, R. (2010). Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*, SoCC '10, pages 143–154, New York, NY, USA. ACM.

[Costa et al., 2011] Costa, P., Pasin, M., Bessani, A. N., and Correia, M. (2011). Byzantine Fault-Tolerant MapReduce: Faults are Not Just Crashes. In *IEEE 3rd International Conference on Cloud Computing Technology and Science (CloudCom 2011)*, Athens, Greece.

[Dean and Ghemawat, 2004] Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *USENIX Symp. on Operating Systems Design and Implementation (OSDI)*.

[DeWitt and Gray, 1992] DeWitt, D. and Gray, J. (1992). Parallel Database Systems: The Future of High Performance Database Systems. *ACM Communications*, 35(6):85–98.

[Dittrich et al., 2010] Dittrich, J., Jindal, A., Kargin, Y., Setty, V., and Schad, J. (2010). Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). In *International Conference on Very Large Data Bases (VLDB)*.

[Duhl and Damon, 1988] Duhl, J. and Damon, C. (1988). A Performance Comparison of Object and Relational Databases Using the Sun Benchmark. In *Conference proceedings on Object-oriented programming systems, languages and applications*, OOPSLA '88, pages 153–163. ACM.

[Durães et al., 2004] Durães, J., Vieira, M., and Madeira, H. (2004). Dependability Benchmarking of Web-Servers. In *Proceedings of 23rd International Conference on Computer Safety, Reliability and Security (Safecomp'2004)*, pages 297–310.

[Eltabakh et al., 2011] Eltabakh, M., Tian, Y., Ozcan, F., Gemulla, R., Krettek, A., and McPherson, J. (2011). CoHadoop: Flexible Data Placement and Its Exploitation in Hadoop. In *International Conference on Very Large Data Bases (VLDB)*.

[Fadika and Govindaraju, 2010] Fadika, Z. and Govindaraju, M. (2010). LEMO-MR: Low Overhead and Elastic MapReduce Implementation Optimized for Memory and CPU-Intensive Applications. In *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*.

[Floratou et al., 2011] Floratou, A., Patel, J., Shekita, E., and Tata, S. (2011). Column-Oriented Storage Techniques for MapReduce. In *International Conference on Very Large Data Bases (VLDB)*.

[Friginal et al., 2011] Friginal, J., de Andres, D., Ruiz, J.-C., and Moraes, R. (2011). Using Dependability Benchmarks to Support ISO/IEC SQuaRE. *IEEE Pacific Rim International Symposium on Dependable Computing*, pages 28–37.

[Frumkin and Van der Wijngaart,] Frumkin, M. and Van der Wijngaart, R. NAS Grid Benchmarks: A Tool for Grid Space Exploration. In *Proceedings*

of 10th IEEE International Symposium on High Performance Distributed Computing, 2001.

[Gray, 1992] Gray, J. (1992). *Benchmark Handbook: For Database and Transaction Processing Systems.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Head et al., 2005] Head, M., Govindaraju, M., Slominski, A., Liu, P., Abu-Ghazaleh, N., van Engelen, R., Chiu, K., and Lewis, M. (2005). A Benchmark Suite for SOAP-based Communication in Grid Web Services. In *Proceedings of the ACM/IEEE Conference of Supercomputing, (SC 2005)*, pages 19–19.

[Herodotou and Babu, 2011] Herodotou, H. and Babu, S. (2011). Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. In *International Conference on Very Large Data Bases (VLDB).*

[Huang et al., 2013] Huang, Q., Yang, C., Liu, K., Xia, J., Xu, C., Li, J., Gui, Z., Sun, M., and Li, Z. (2013). Evaluating open-source cloud computing solutions for geosciences. *Computers & Geosciences*, 59:41–52.

[Huang et al., 2010] Huang, S., Huang, J., Dai, J., Xie, T., and Huang, B. (2010). The HiBench Benchmark Suite: Characterization of the MapReduce-Based Data Analysis. In *IEEE International Conference on Data Engineering Workshops (ICDEW).*

[Huppler, 2012] Huppler, K. (2012). Benchmarking with your head in the cloud. In *Proceedings of the Third TPC Technology conference on Topics in Performance Evaluation, Measurement and Characterization*, TPCTC'11, pages 97–110.

[Iosup, 2013] Iosup, A. (2013). IaaS Cloud Benchmarking: Approaches, Challenges, and Experience. In *Proceedings of the 2013 international workshop on Hot topics in cloud services*, HotTopiCS '13, pages 1–2, New York, NY, USA. ACM.

[Iosup et al., 2011] Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2011). Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel Distributed Systems*, 22(6):931–945.

[Isard et al., 2009] Isard, M., Prabhakaran, V., Currey, J., Wieder, U., Talwar, K., and Goldberg, A. (2009). Quincy: Fair Scheduling for Distributed Computing Clusters. In *Symposium on Operating Systems Principles (SOSP).*

[Jackson et al., 2010] Jackson, K. R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H. J., and Wright, N. J. (2010). Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 159–168, Washington, DC, USA. IEEE Computer Society.

[Jaleel et al., 2006] Jaleel, A., Mattina, M., and Jacob, B. (2006). Last Level Cache (LLC) Performance of Data Mining Workloads on a CMP - A Case Study of Parallel Bioinformatics Workloads. In *The Twelfth International Symposium on High-Performance Computer Architecture*, pages 88–98.

[Jannach et al., 2010] Jannach, D., Zanker, M., Felfernig, A., and Friedrich, G. (2010). *Recommender Systems: An Introduction.*

[Jin et al., 2012] Jin, H., Yang, X., Sun, X.-H., and Raicu, I. (2012). ADAPT: Availability-Aware MapReduce Data Placement in Non-Dedicated Distributed Computing Environment. In *IEEE Int. Conf. on Distributed Computing Systems (ICDCS).*

[Kanoun et al., 2005] Kanoun, K., Crouzet, Y., Kalakech, A., Rugina, A.-E., and Rumeau, P. (2005). Benchmarking the dependability of windows and linux using postmark™ workloads. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, ISSRE '05, pages 11–20.

[Kim et al., 2008] Kim, K., Jeon, K., Han, H., Kim, S.-g., Jung, H., and Yeom, H. Y. (2008). MRBench: A Benchmark for MapReduce Framework. In *IEEE Int. Conf. on Parallel and Distributed Systems (ICPADS).*

[Ko et al., 2010] Ko, S. Y., Hoque, I., Cho, B., and Gupta, I. (2010). Making Cloud Intermediate Data Fault-Tolerant. In *ACM Symp. on Cloud computing (SoCC).*

[Koopman et al., 1997] Koopman, P., Sung, J., Dingman, C., Siewiorek, D., and Marz, T. (1997). Comparing Operating Systems Using Robustness Benchmarks. In *Proceedings of the 16th Symposium on Reliable Distributed Systems*, SRDS '97.

[Kossmann et al., 2010] Kossmann, D., Kraska, T., and Loesing, S. (2010). An Evaluation of Alternative Architectures for Transaction Processing in the Cloud. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 579–590, New York, NY, USA. ACM.

[Laprie, 1995] Laprie, J.-C. (1995). Dependable Computing and Fault-Tolerance: Concepts and Terminology. In *25th International Symposium on Fault-Tolerant Computing*.

[Laranjeiro et al., 2008] Laranjeiro, N., Canelas, S., and Vieira, M. (2008). wsrbench: An On-Line Tool for Robustness Benchmarking. In *Proceedings of the 2008 IEEE International Conference on Services Computing - Volume 2*, SCC '08, pages 187–194.

[Lee et al., 2009] Lee, J. Y., Lee, J. W., Cheun, D. W., and Kim, S. D. (2009). A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. *ACIS International Conference on Software Engineering Research, Management and Applications*, 0:261–266.

[Li et al., 2005] Li, M.-L., Sasanka, R., Adve, S., Chen, Y.-K., and Debes, E. (2005). The ALPBench Benchmark Suite for Complex Multimedia Applications. In *Proceedings of the IEEE International Workload Characterization Symposium*, pages 34–45.

[Lin et al., 2010] Lin, H., Ma, X., Archuleta, J., Feng, W.-c., Gardner, M., and Zhang, Z. (2010). MOON: MapReduce On Opportunistic eNvironments. In *ACM Int. Symp. on High Performance Distributed Computing (HPDC)*.

[Liu and Orban, 2011] Liu, H. and Orban, D. (2011). Cloud MapReduce: A MapReduce Implementation on Top of a Cloud Operating System. In *IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGRID)*.

[Lu, 2010] Lu, M.-Y. (2010). HadoopToSQL. In *EuroSys Conference*.

[Mauro et al., 2004] Mauro, J., Zhu, J., and Pramanick, I. (2004). The System Recovery Benchmark. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, PRDC '04, pages 271–280.

[Mehrotra et al., 2012] Mehrotra, P., Djomehri, J., Heistand, S., Hood, R., Jin, H., Lazanoff, A., Saini, S., and Biswas, R. (2012). Performance evaluation of Amazon EC2 for NASA HPC applications. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, ScienceCloud '12, pages 41–50, New York, NY, USA. ACM.

[Menasce, 2002] Menasce, D. (2002). TPC-W: a benchmark for e-commerce. *IEEE Internet Computing*, 6(3):83–87.

[Mizouni et al., 2011] Mizouni, R., Serhani, M., Dssouli, R., Benharref, A., and Taleb, I. (2011). Performance Evaluation of Mobile Web Services. In *Ninth IEEE European Conference on Web Services (ECOWS)*, pages 184–191.

[Neumeyer et al., 2010] Neumeyer, L., Robbins, B., Nair, A., and Kesari, A. (2010). S4: Distributed Stream Computing Platform. In *IEEE International Conference on Data Mining Workshops (ICDMW), 2010*, pages 170–177.

[Nicola et al., 2007] Nicola, M., Kogan, I., and Schiefer, B. (2007). An XML Transaction Processing Benchmark. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 937–948, New York, NY, USA. ACM.

[Nishtala et al., 2013] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., Mcelroy, R., Paleczny, M., Peek, D., Saab, P., Stafford, D., Tung, T., and Venkataramani, V. (2013). Scaling Memcache at Facebook. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, NSDI'13, pages 385–398.

[Nurmi et al., 2009] Nurmi, D., Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., and Zagorodnov, D. (2009). The Eucalyptus Open-Source Cloud-Computing System. In *9th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 124–131.

[Ostermann et al., 2010] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., and Epema, D. (2010). A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, pages 115–131.

[Pavlo et al., 2009] Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., and Stonebraker, M. (2009). A Comparison of Approaches to Large-Scale Data Analysis. In *ACM International Conference on Management of Data (SIGMOD)*.

[Pham et al., 2012] Pham, C., Cao, P., Kalbarczyk, Z., and Iyer, R. (2012). Toward a high availability cloud: Techniques and challenges. In *Dependable Systems and Networks Workshops (DSN-W), 2012 IEEE/IFIP 42nd International Conference on*, pages 1–6.

[Rimal et al., 2009] Rimal, B., Choi, E., and Lumb, I. (2009). A Taxonomy and Survey of Cloud Computing Systems. In *Fifth International Joint Conference on INC, IMS and IDC, 2009. NCM '09.*, pages 44–51.

[Rosenberg et al., 2006] Rosenberg, F., Platzer, C., and Dustdar, S. (2006). Bootstrapping Performance and Dependability Attributes of Web Services. In *International Conference on Web Services, (ICWS '06)*, pages 205–212.

[Sangroya et al., 2012a] Sangroya, A., Serrano, D., and Bouchenak, S. (2012a). Benchmarking Dependability of MapReduce Systems. In *IEEE Int. Symposium on Reliable Distributed Systems (SRDS).*

[Sangroya et al., 2012b] Sangroya, A., Serrano, D., and Bouchenak, S. (2012b). MRBS: A Comprehensive MapReduce Benchmark Suite. Research Report RR-LIG-024, LIG, Grenoble, France.

[Schatz, 2009] Schatz, M. C. (2009). CloudBurst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics.*

[Schmidt et al., 2002] Schmidt, A., Waas, F., Kersten, M., Carey, M. J., Manolescu, I., and Busse, R. (2002). XMark: A Benchmark for XML Data Management. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 974–985. VLDB Endowment.

[Schmidt et al., 2001] Schmidt, A., Waas, F., Kersten, M., Florescu, D., Carey, M. J., Manolescu, I., and Busse, R. (2001). Why and How to Benchmark XML Databases. *SIGMOD Rec.*, 30(3):27–32.

[Sempolinski and Thain, 2010] Sempolinski, P. and Thain, D. (2010). A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus. In *Second IEEE International Conference onCloud Computing Technology and Science (CloudCom)*, pages 417–426.

[Serrano et al., 2013] Serrano, D., Bouchenak, S., Kouki, Y., Ledoux, T., Lejeune, J., Sopena, J., Arantes, L., and Sens, P. (2013). Towards QoS-Oriented SLA Guarantees for Online Cloud Services. *IEEE International Symposium on Cluster Computing and the Grid*, 0:50–57.

[Stantchev, 2009] Stantchev, V. (2009). Performance Evaluation of Cloud Computing Offerings. In *Third International Conference on Advanced Engineering Computing and Applications in Sciences, (ADVCOMP '09)*, pages 187–192.

[Tsai et al., 2011] Tsai, W., Huang, Y., and Shao, Q. (2011). Testing the Scalability of SaaS Applications. In *IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2011*, pages 1–4.

[Verma et al., 2011] Verma, A., Cherkasova, L., and Campbell, R. H. (2011). Resource Provisioning Framework for MapReduce Jobs with Performance Goals. In *Middleware Conference*.

[Vieira et al., 2007] Vieira, M., Laranjeiro, N., and Madeira, H. (2007). Benchmarking the Robustness of Web Services. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, PRDC '07, pages 322–329.

[Vieira and Madeira, 2003] Vieira, M. and Madeira, H. (2003). A Dependability Benchmark for OLTP Application Environments. In *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pages 742–753.

[Voorsluys et al., 2009] Voorsluys, W., Broberg, J., Venugopal, S., and Buyya, R. (2009). Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 254–265, Berlin, Heidelberg. Springer-Verlag.

[White, 2009] White, T. (2009). *Hadoop: The Definitive Guide*. O'Reilly Media. http://hadoop.apache.org/.

[Wickramage and Weerawarana, 2005] Wickramage, N. and Weerawarana, S. (2005). A benchmark for Web Service Frameworks. In *IEEE International Conference on Services Computing*, volume 1, pages 233–240 vol.1.

[Woo et al., 1995] Woo, S. C., Ohara, M., Torrie, E., Singh, J. P., and Gupta, A. (1995). The SPLASH-2 Programs: Characterization and Methodological Considerations. *SIGARCH Computer Architecture News*, 23(2):24–36.

[Yao et al., 2004] Yao, B., Ozsu, M., and Khandelwal, N. (2004). XBench Benchmark and Performance Testing of XML DBMSs. In *Proceedings of 20th International Conference on Data Engineering, 2004*, pages 621–632.

[Zaharia et al., 2010] Zaharia, M., Borthakur, D., Sarma, J. S., Elmeleegy, K., Shenker, S., and Stoica, I. (2010). Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling. In *EuroSys Conference*.

[Zaharia et al., 2008] Zaharia, M., Konwinski, A., Joseph, A. D., Katz, R., and Stoica, I. (2008). Improving MapReduce Performance in Heterogeneous Environments. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[Zheng et al., 2012] Zheng, Q., Chen, H., Wang, Y., Duan, J., and Huang, Z. (2012). COSBench: A Benchmark Tool for Cloud Object Storage Services. *IEEE Fifth International Conference on Cloud Computing*.

[Zhu et al., 2006] Zhu, L., Gorton, I., Liu, Y., and Bui, N. B. (2006). Model Driven Benchmark Generation for Web Services. In *Proceedings of the 2006 International workshop on Service-oriented software engineering*, SOSE '06, pages 33–39. ACM.