Modeling and control of cloud services : application to MapReduce performance and dependability

Mihaly Berekmeri

UNIVERSITÉ GRENOBLE ALPES

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Automatique-Productique**

Arrêté ministériel : 7 août 2006

Présentée par

Mihály BEREKMÉRI

Thèse dirigée par **Sara BOUCHENAK** et
codirigée par **Bogdan ROBU**

préparée au sein du **GIPSA-lab et LIRIS**
dans **l'École Doctorale EEATS**

La Modélisation et le Contrôle des services BigData: Application à la Performance et la Fiabilité de MapReduce

Thèse soutenue publiquement le **18 Novembre 2015**,
devant le jury composé de :

Monsieur Karl-Erik ÅRZÉN
Professeur**,** Lund University, Rapporteur
Madame Karama KANOUN
Directeur de Recherche, LAAS-CNRS, Rapporteur
Madame Lydia Y.CHEN
Docteur, IBM, Membre
Monsieur Daniel SIMON
Chargé de Recherche, INRIA, Membre
Monsieur Nicolas MARCHAND
Directeur de Recherche, CNRS, Président
Madame Sara BOUCHENAK
Professeur, INSA Lyon, Membre
Monsieur Bogdan ROBU
Maîtres de conférences, Université Grenoble Alpes, Membre

UNIVERSITÉ GRENOBLE ALPES

ÉCOLE DOCTORALE EEATS

Electronique, Electrotechnique, Automatique, Traitement du signal

T H È S E

pour obtenir le titre de

docteur en sciences

de l'Université Grenoble Alpes

Mention : AUTOMATIQUE-PRODUCTIQUE

Présentée et soutenue par

Mihály Berekméri

Modeling and control of cloud services

Application to MapReduce performance and dependability

Thèse dirigée par Sara BOUCHENAK

préparée au Département Automatique du GIPSA-Lab et LIRIS

soutenue le 18 Novembre 2015

Jury :

Rapporteurs :	Karl-Erik ÅRZÉN	-	Lund University
	Karama KANOUN	-	CNRS
Directrice :	Sara BOUCHENAK	-	*INSA Lyon*
Co-Directeur :	Bogdan ROBU	-	Grenoble Alpes University
Examinateur :	Lydia Y. Chen	-	IBM Zurich
	Daniel SIMON	-	INRIA
	Nicolas MARCHAND	-	CNRS

Modeling and control of cloud services
Application to MapReduce performance and dependability

Mihaly Berekmeri
GIPSA-lab, LIRIS, University of Grenoble Alpes
Grenoble, France
mihaly.berekmeri@gipsa-lab.fr

Abstract: The amount of raw data produced by everything from our mobile phones, tablets, computers to our smart watches brings novel challenges in data storage and analysis. Many solutions have arisen in the industry to treat these large quantities of raw data, the most popular being the MapReduce framework. However, while the deployment complexity of such computing systems is steadily increasing, continuous availability and fast response times are still the expected norm. Furthermore, with the advent of virtualization and cloud solutions, the environments where these systems need to run is becoming more and more dynamic. Therefore ensuring performance and dependability constraints of a MapReduce service still poses significant challenges. In this thesis we address this problematic of guaranteeing the performance and availability of MapReduce based cloud services, taking an approach based on control theory. We develop the first dynamic models of a MapReduce service running a concurrent workload. Furthermore, we develop several coarse-grained control laws to ensure different quality of service objectives. First, classical feedback and feedforward controllers are developed to guarantee service performance. To further adapt our controllers to the cloud, such as minimizing the number of reconfigurations and costs, a novel event-based control architecture is introduced for performance management. Finally we develop the optimal control architecture *MR-Ctrl*, which is the first solution to provide guarantees in terms of both performance *and* dependability for MapReduce systems, meanwhile keeping cost at a minimum. All the modeling and control approaches are evaluated both in simulation and with the use of MRBS, a comprehensive benchmark suite for evaluating the performance and dependability of MapReduce systems. Validation experiments were run in a real 60 node Hadoop MapReduce cluster, running a data intensive Business Intelligence workload. Our experiments show that the proposed techniques can successfully guarantee performance and dependability constraints.

Keywords: Modeling; Control; MapReduce; Performance; Dependability; Feedback; Feedfoward; Optimal;

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisors, professors Sara Bouchenak and Bogdan Robu, for their continuous guidance during my Ph.D study, for their patience and kindness. Without their support this work would not have been possible.

I especially wish to thank professor Nicolas Marchand for always keeping his door open for my questions and for his continuous mentorship during the course of this work.

I want to thank jury members Karl-Erik Årzén, Karama Kanoun, Lydia Y. Chen and Daniel Simon for their valuable insights and suggestions to improve the quality of this manuscript.

I am also grateful for Damian Serrano Garcia for all his help with MapReduce and MRBS.

I would like to extend my sincerest thanks to professors Haller Piroska and Márton Lőrinc for introducing me into the field of research during my undergraduate studies. Their kindness and continuous support has been invaluable.

Furthermore, allow me to express my appreciation towards all my colleagues and friends at GIPSA-lab for creating an environment where I felt at home and for always providing a helping hand whenever I was in need of it.

On a more personal level, my thoughts go out to my parents and my brother who continuously supported me throughout the years.

Last but not least, I would like to thank my love Anna for always being there for me with great love and care.

This thesis is also the fruit of all their labour.

Contents

Table of acronyms **ix**

1 Introduction **1**

1.1 General context 1

1.2 Thesis contributions 3

1.3 Thesis outline 4

2 Background and Definitions **7**

2.1 Cloud computing 7

2.2 Service Level Agreements 8

2.3 MapReduce 8

2.4 The MapReduce Benchmark Suite (MRBS) 10

2.5 Definitions 11

2.6 MAPE-k loop vs Feedback control loop 12

2.7 Background on control theory for application to computing systems 13

3 Related Work **19**

3.1 Guaranteeing Performance in the Cloud 19

3.2 Mapreduce performance modelling 20

3.3 Controlling MapReduce Performance 22

3.4 Improving the MapReduce Framework 22

3.5 MapReduce Benchmarking and Simulation 24

3.6 Feedback Control of Computing Systems 24

3.7 Discussion 25

4 Motivations and Objectives **27**

4.1 Motivations 27

4.2 Objectives 32

4.3 Summary 33

5 MapReduce Performance and Availability Modelling **35**

5.1 Capturing dynamic behaviour of software systems 36

5.2 Building the MapReduce dynamic model 37

5.3 MapReduce Performance model 39

5.4 MapReduce Availability model 46

5.5 MapReduce Performance and Dependability model 47

5.6 Model identification 50

5.7 Summary 53

6 Control of MapReduce Performance and Availability **55**

6.1 Classical control 56

6.2 Event-based control 61

6.3 Constrained optimal control: *MR-Ctrl* 66

6.4 Summary 72

7 Experimental Evaluation **73**

7.1 Experimental MapReduce Environment 74

7.2 Model experimental validation 78

7.3 Control experimental validation 85

7.4 Summary 97

8 Conclusions and Perspectives **99**

8.1 Conclusions 99

8.2 Perspectives 102

A Grid5000 basics **103**

B Using the MapReduce Benchmark Suite **105**

References **112**

List of Figures

2.1 MapReduce Architecture (Hadoop 1^{st} version) 9

2.2 MapReduce word count example 10

2.3 MRBS architecture 11

2.4 MAPE-k adaptation loop 12

2.5 Feedback control loop 13

2.6 Fixed interval sampling of a continuous signal 14

4.1 Cloud service management concerns by CIOs 28

4.2 Effects of workload variation, #Nodes=20 29

4.3 Effects of cluster size variation, #Clients=10 30

4.4 Effects of admission control level variation, #Nodes=20, #Clients=10 31

5.1 System model with specified inputs/outputs 39

5.2 Effects of workload variation, #MC=10, #Nodes=20 40

5.3 Effects of cluster size variation, #MC=5, #Clients=10 41

5.4 Effects of MC variation,#Nodes=20, #Clients=10 42

5.5 MapReduce performance model 45

5.6 MapReduce availability model 47

5.7 General MIMO state space model structure 49

5.8 General identification procedure 50

6.1 Classical control architecture 56

6.2 Difference between time-based and event-based control instants 62

6.3 Event-based control architecture 63

6.4 *MR-Ctrl* : Optimal control architecture 67

7.1 Intuitive view of the experimental setup 74

7.2 Detailed view of the experimental setup 75

7.3 Identification of the undisturbed system. It predicts the effects of cluster size reconfigurations on performance 78

7.4 Identification of the disturbance model. It captures the effect of workload size variations on performance 79

7.5 Performance model validation [17]. It predicts the effects of cluster and workload size variation on performance 80

7.6 Availability model validation with $N = 20, C = 10$. It predicts the effects of max client level variations on availability 81

7.7 MIMO model validation for workload variation, #MC=5, #Nodes=20 82

7.8 MIMO model validation for cluster size variation, #MC=5, #Clients=10 83

7.9 Classical PI feedback control - experimental evaluation 87

7.10 Event-based PI feedback control - experimental evaluation 88

7.11 Classical feedfoward control - experimental evaluation 89

7.12 Event-based feedforward control - experimental evaluation 91

7.13 Modified event-based feedforward control - experimental evaluation 92

7.14 Event-based feedback and modified feedforward control - evaluation in simulation 93

7.15 Performance guardian and Availability guardian control scenarios - evaluation in simulation 95

7.16 Availability guardian control scenario - experimental evaluation 96

Table of acronyms

avg	*average*
BI	*business intelligence*
CI	*client interaction*
CIO	*chief information officer*
FOPDT	*first order plus dead-time*
HDFS	*hadoop distributed file system*
I	*identity matrix*
IaaS	*infrastructure as a service*
LTI	*linear time invariant*
MAPE-k	*monitor analyse plan execute knowledge*
MC	*max clients level*
MIMO	*multi-input, multi-output*
MRBS	*mapreduce benchkmark suite*
PaaS	*platform as a service*
PID	*proportional-integral-derivative control*
SaaS	*software as a service*
SED	*stream editor*
SISO	*single-input, single-output*
SLA	*service level agreement*
SLO	*service level objective*
SSH	*secure shell protocol*
SQL	*structured query language*
TPC	*transaction processing council*

Chapter 1

Introduction

1.1 General context

We are at the dawn of a data and computing revolution. The amount of raw data produced by everything from our mobile phones, tablets, computers to our smart watches is increasing exponentially. As a result companies face novel and growing challenges in data storage and analysis. The sheer amount of data available is asking for a shift of perspective from the traditional database approaches to platforms capable of handling petabytes of unstructured information available for tasks such personalized advertising, advanced data mining or classification.

One of the most popular of such current platforms is the MapReduce framework which is one of the currently most utilised programming paradigms in use for parallel, distributed computations over large amounts of data. Nowadays MapReduce is backed by the largest BigData industry leaders such as Google, Yahoo, Facebook and Amazon. For example, Google has more than 100.000 MapReduce jobs executed every day [20] , Yahoo has more the 40.000 computers running MapReduce jobs, Linkedin evaluates around 120 billion relationships per day [69] using MapReduce while, Facebooks largest Hadoop MapReduce contains more than a 100 petabytes of data.

At the same time cloud computing, the next milestone of IT evolution, is becoming a more and more attractive option for many companies. Its promise of "unlimited" storage and processing capabilities together with its pay as you go approach is proving an enticing solution. It allows easy access to a group of shared computing resources in the form of an internet service. These computing resources can vary from hardware, platform, software to storage. Therefore customers can start up with low cost computing profile and easily scale up or down as necessary as their business evolves. Nevertheless, while current state of the art commercial MapReduce services such as Amazon EMR [5] and Microsoft HDInsight [48] offer solutions for quick and cost-effective data processing they don't provide any guarantees in terms of application performance and dependability. While elasticity mechanism are given, they are not completely automatic and several important scaling decisions, such as selecting the scaling thresholds, are left up to the service user.

Moreover, resource provisioning for deadline management in the cloud is further made difficult because of the shared hardware resource architecture, where interference and concurrency issues may arise frequently and workloads fluctuate over time. Furthermore, as cloud

providers desire to maximise the resource utilisation, they have mechanisms for the dynamic reallocation of unused resources which further adds to the variability of system performance. So even with the same workload and resource amount, an application performance may vary depending on how noisy neighbouring applications are. In the meantime, for most businesses of course, missing deadlines, results in financial losses. In some cases the service unavailability may costs up to 100.000$ per minute, as is the case of an on-line brokerage industry [23].

Meanwhile, ensuring performance and dependability of MapReduce systems is not trivial. Although the framework hides the complexities of parallelism from the service users, deploying an efficient MapReduce implementation poses multiple challenges. MapReduce's ad-hoc configuration and provisioning require a high level of expertise to tune [81]. Furthermore many factors have been identified that negatively influence the performance of MapReduce jobs: CPU, input/output and network skews [73], hardware and software failures [64], node homogeneity assumption not holding up [87], and bursty workloads [18].

As results lots of research is being done in the computing community on improving the performance, availability of complex computing systems such as MapReduce. Extensive research has been conducted already to improve dependability or performance of MapReduce [12, 11, 88] by changing the behaviour and algorithms of the MapReduce framework itself. Although these solutions improve upon how MapReduce works no guarantees are provided in term of performance and availability. Although several solutions for performance modelling [77, 74, 39] and control [15, 76] can be found in the literature, there are no works to provide concurrent guarantees in terms of both dependability and performance for a concurrent workload. Furthermore, there are still many unanswered questions. Such as how to use the classical techniques to guarantee performance to provide on-line assurances when the systems are designed with only partial knowledge of their runtime environment? Because of the unpredictability of the new environments, traditional adaptation approaches become increasingly difficult to use. Therefore more and more attention is given to approaches used in different fields for tackling complex systems.

The most prominent of these are the feedback control solutions coming from the field of control theory [32, 54], which has been providing answers to these questions for physical systems for several decades now. The advantages of control theory are that it can provide a solid mathematical basis for synthesizing feedback control loops, for handling safely complexity and for having theoretically guaranteed results. However, applying control theory to computing systems is not straight forward. Contrary to physical systems there are no physics governing algorithms, software. How do we use the classical techniques to build models of MapReduce cloud systems? What do we measure? How do we chose our actuators? How to manage the trade-off between performance, dependability and cost? These are just a few of the questions that are still not fully addressed [25].

1.2 Thesis contributions

The principle contributions of the thesis are the modelling and control of the performance and dependability of MapReduce cloud software systems. In the following a summary of these results is presented.
The main theoretical results of the thesis are the following:

1. The first contribution of the thesis is the design, implementation and validation of the first models that can capture the dynamic performance and availability of a MapReduce cluster running a data intensive, concurrent workload.
2. The second contribution is the design, implementation and evaluation of multiple control laws capable of ensuring the performance and the availability of a MapReduce cluster. First, a control architecture is developed that can guarantee the MapReduce performance through cluster scaling, based on a classical time based PI and feedforward controllers. We further improve upon the previous time based control architecture by adapting it to the cloud scenario through event-based control techniques. Finally, the optimal control framework called *MR-Ctrl* is introduced, that can guarantee at the same time both performance and availability of a MapReduce service, while explicitly minimising control costs.

The main technical result of the thesis is the construction of experimental environment that allows for the easy development and testing of different modeling and control strategies of a real MapReduce service, from any local computer running Matlab. This involved the writing all of the low level actuator and sensor scripts, the communication interface between Matlab and the remote MapReduce cluster. Moreover, an automatic experiment deployment framework was implemented that allows for unsupervised experimental runs.

Publication list:

- International conference papers:
  - "Application du contrôle pour garantir la performance des systèmes Big Data" M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand, B. Robu. Conférence en Parallélisme, Architecture et Système (ComPAS) 2014, Neuchâtel, Switzerland, April 22-25, 2014
  - "A Control Approach for Performance of Big Data Systems".
    M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand and B. Robu.
    Proc. of the 19th World Congress of the International Federation of Automatic Control, IFAC 2014, Cape-Town, South Africa, August 24-29, 2014. pp 152-157
- Journal papers:
  - "Feedback Autonomic Provisioning for Guaranteeing Performance in MapReduce Systems".M. Berekmeri, D. Serrano, S. Bouchenak, N. Marchand and B. Robu.
    IEEE Transactions on Cloud Computing (accepted)

1.3 Thesis outline

This thesis is organised in six main parts. The first chapter presents the context and the thesis contributions. The second chapter contains an introduction to the concepts utilised in the manuscript. The third chapter is dedicated to establishing the context of our work through a detailed analysis of the speciality literature. The fourth chapter details our motivations and the objectives of the thesis are defined. The fifth chapter presents our proposed dynamic models for the performance and/or availability of MapReduce systems. The control architectures developed to ensure the performance and availability of MapReduce systems are described in Chapter six. Chapter seven contains the experimental evaluation of all the presented modelling and control solutions. Finally, Chapter eight draws the conclusions and presents our ideas for future work.

In the following a short description of each chapter is provided.

Chapter 1 - Introduction presents the context of our work and the thesis contributions.

Chapter 2 - Background and Definitions contains the background of the control theoretical and computing concepts required for the understanding of the manuscript. First we introduce the utilised computing technologies such as Cloud computing, MapReduce, Hadoop and MRBS. To bridge the on-line adaptation methods traditionally used in computer science and control engineering, a detailed comparison of the MAPE-k and feedback loop is presented. In addition, the control theoretical concepts used in this thesis are shortly presented to facilitate the readability by those with a computer science background.

Chapter 3 - Related work. In this chapter we take a thorough look at the related work to determine the current challenges and limitations in the modelling, performance and dependability control of cloud systems, with a special focus on the MapReduce framework. At first, a broader view of the issues regarding the performance control of cloud services is presented. Then, we focus on the state of the art methods for the performance modelling and control of MapReduce systems. We also take into account the large research effort in improving the performance and dependability of MapReduce systems. In addition, a brief overview of the existing MapReduce benchmarking and simulation approaches is presented as well. We also consider the state of feedback control usage in case of computing systems. Finally, the challenges and limitations of existing approaches is discussed and the open issues are highlighted.

Chapter 4 - Motivation and Objectives. This chapter deals with the motivations of our work. We provide a detailed analysis of current consumer concerns with cloud systems and examine the effects of workload variations on the performance and availability of a MapReduce cluster. Moreover, the effects of possible corrective control knobs, such as cluster scaling and admission control, are experimentally evaluated as well. Finally, concluding on the context and motivations part of the thesis, the objectives are defined.

Chapter 5 - MapReduce Performance and Availability Modelling gathers the steps taken to model the dynamic behaviour of a MapReduce cluster. In the beginning, we introduce a methodology for assigning a measurable dynamics to such software systems and we perform a preliminary analysis of MapReduce dynamics. Based on the previous analysis we propose a general performance and dependability model structure for MapReduce systems. Furthermore, we gradually explain the techniques used to identify the relationships between each model input and output. First, a performance model that captures the connection between the cluster, workload sizes and the average runtime is developed. Second, the correlation between availability and the max clients level is presented. The two previous models are combined and the general multi-input, multi-output (MIMO) model of the MapReduce system is proposed. Finally, we present the off-line and on-line methodologies used to identify the parameters of the developed models.

Chapter 6 - Control of MapReduce Performance and Availability. In this chapter the on-line control algorithms for ensuring a MapReduce clusters performance and dependability are developed. First, a control architecture that ensures MapReduce performance through cluster scaling, based on a classical, time based PI and feedforward controllers, is introduced. Following this, we further adapt the previous time based control architecture to the cloud environment through event-based techniques by elaborating an event-based PI and feedforward control architecture. Finally, the optimal control framework *MR-Ctrl* is presented, that ensures at the same time both the performance and availability of a MapReduce service, meanwhile explicitly minimising control costs.

Chapter 7 - Experimental Results. The MapReduce dynamic models developed in Chapter 5 and on-line control algorithms elaborated in Chapter 6 are evaluated in this chapter. First, a detailed presentation of the developed experimental setup, where all the validation experiments have been run, is given. Second, using this testbed, we experimentally validate the models developed in Chapter 5. Finally, we provide the numeric and experimental evaluation of the control algorithms elaborated in Chapter 6.

Chapter 8 - Conclusions and perspectives. The chapter rounds up our work, by drawing the conclusions and detailing the perspectives for mid-term and long-term future works.

Chapter 2

Background and Definitions

Contents

2.1 Cloud computing . . . 7
2.2 Service Level Agreements . . . 8
2.3 MapReduce . . . 8
2.4 The MapReduce Benchmark Suite (MRBS) . . . 10
2.5 Definitions . . . 11
2.6 MAPE-k loop vs Feedback control loop . . . 12
2.7 Background on control theory for application to computing systems 13
2.7.1 White-box vs Black-box modeling . . . 13
2.7.2 Discrete time signal . . . 14
2.7.3 Difference equations . . . 15
2.7.4 Z transform . . . 15
2.7.5 Discrete time transfer function . . . 16
2.7.6 First order plus dead-time models - FOPDT . . . 17
2.7.7 Linear vs Non-linear Systems . . . 17
2.7.8 System stability, performance and robustness . . . 18

This chapter presents the necessary background for the understanding of this manuscript. First the utilised computing technologies, such as Cloud computing, MapReduce, Hadoop and MRBS, are introduced. Then we perform a detailed comparison between the MAPE-k and feedback loops, the on-line adaptation mechanism used in software engineering and respectively control theoretic domains. Finally, a short introduction into the control theoretical concepts used in the manuscript is given, to facilitate the understanding of the document by those without a control theoretical background.

2.1 Cloud computing

Cloud Computing is the next milestone of IT evolution. It allows easy access to a group of shared computing resources in the form of an internet service. These computing resources can vary from hardware, platform, software to storage. The service comes with metering capabilities and it is built upon the pay as you go model. Therefore customers can start up

with a low cost computing profile and easily scale up or down when necessary, as their business evolves. This rapid elasticity (scalability) is one of the key advantages of Cloud computing.

Cloud services can be classified into three categories based on the level of abstraction offered:

1. Infrastructure as a service (IaaS) is the most wide spread service model at the moment and it consists of the offer of computers (physical or virtual) as a service, for example Amazon EC2, Google Compute Engine, Windows Azure Virtual Machines.

2. Platform as a service (PaaS) models offer a complete computing framework such as programming models and languages, operating systems. The key advantage here being that users of this platform don't have to worry about buying, maintaining the necessary hardware, software layers for these frameworks. Many companies already offer such services for example the Google App Engine, Amazon Elastic Beanstalk.

3. Software as a service (SaaS) provides an environment for running end user applications in the cloud, for example Google Apps, Microsoft Office 365.

2.2 Service Level Agreements

Service Level Agreements (SLAs) are also a relatively a fresh area of research in cloud systems, see [16, 27, 66]. SLA is a contract negotiated between clients and their service provider [20], where service performance is part of the agreement. The SLA can specify service level objectives (SLOs) such as the maximum response time to be guaranteed by the provider. Although current cloud solutions do not provide guarantees regarding service performance and dependability, we believe that more and more customers will be interested on having such guarantees and that those service level providers that can supply them, will gain a competitive advantage. For more details about this issue see European projects such as the MyCloud European project [58] or HARNESS [57], which propose PAAS services that provision themselves based on SLA.

2.3 MapReduce

The main objectives of Big Data Clouds are to capture, store, analyse and manipulate large and complex amounts of unstructured data. MapReduce is one of the currently most used programming paradigms developed for parallel, distributed computations over large amounts of data. The MapReduce programming paradigm was introduced by J. Dean and S. Ghemawat, from Google, in 2004 for large scale unstructured data processing. It has a wide range of applicability, ranging from log analysis, data mining, web search engines, scientific computing to business intelligence.

One of the most important paradigm shifts introduced by MapReduce is to take the computation to the data, instead of transferring the data to the computing nodes. Its success also lies in its simplicity, scalability and fault-tolerance. Fault tolerance was a highly important factor in the design of MapReduce, as the cluster of computers on which MapReduce is deployed is usually made up of cheap commodity computers, instead of dedicated servers. Furthermore, the framework automatically takes care of data partitioning, consistency, replication, task distribution, scheduling and load balancing. Therefore when designing a new job, the programmer's focus can be on the task at hand and not on worrying about the messy overhead associated with most of the other parallel processing algorithms. When designing a MapReduce job, the developer has to implement only two functions: the Map function and the Reduce function, which leaves much less room for programming errors as well.

As we can see in Figure 2.1 MapReduce's initial implementation is based on a master-slave architecture. The central controller is made up of the JobTracker and is in charge of task scheduling, monitoring and resource management. The slaves in this architecture are the TaskTrackers. They are in charge of starting and monitoring local mapper and reducer processes and regularly reporting to the JobTracker. The input data is automatically partitioned into smaller data chunks and each of these is processed in parallel by a Mapper. The intermediate data produced by the mapping stage is grouped together and the final output to the query is produced in reduce stage.

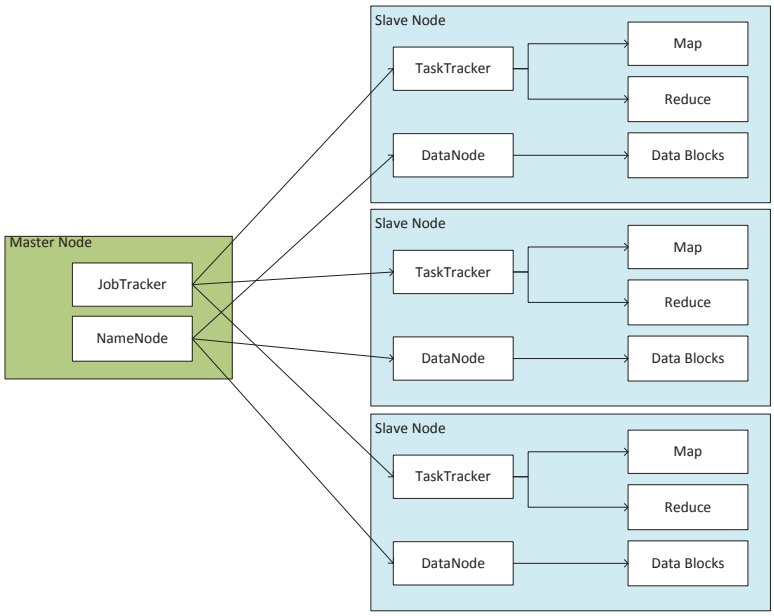


Figure 2.1: MapReduce Architecture (Hadoop 1^{st} version)

To get a better understanding of how MapReduce works a simple example is solved in detail in Figure 2.2. Our task is to count the occurrence of each word in a file. First the framework

splits up the file line by line and each line is sent to a different computing node called Mapper. Each of these Mappers processes their line in parallel and for each word a <key,value> pair is generated. In our case the key is the word itself and the value assigned is always 1. In the shuffling phase the <key,value> pairs are grouped together at the reducers, based on their respective keys. Namely, every occurrence of the word is transferred to a node that simply sums up the values associated to the same keys. Which in fact gives us the occurrence number of the word. The most used open source implementation of the MapReduce programming

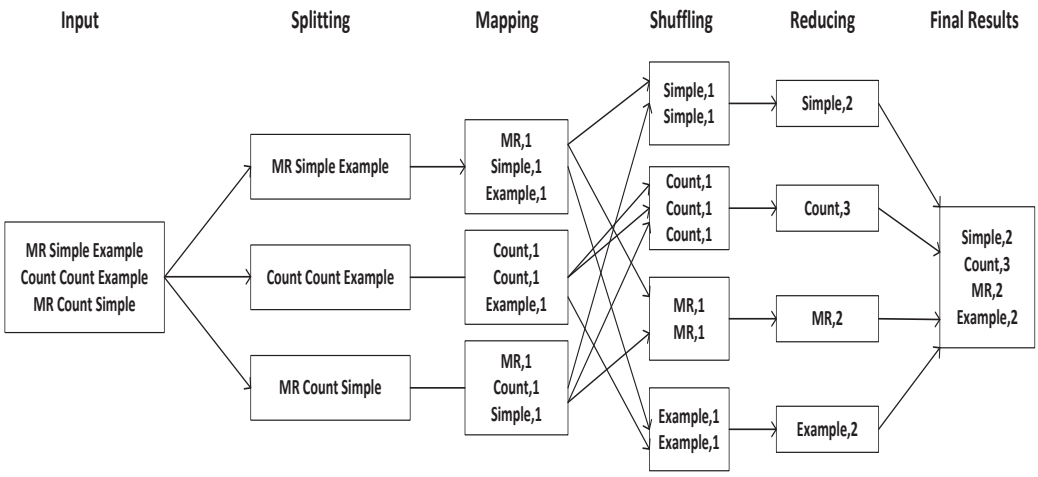


Figure 2.2: MapReduce word count example

model is Hadoop [31]. It is composed of the Hadoop kernel, the Hadoop Distributed Filesystem (HDFS) and the MapReduce engine. Hadoop's HDFS and MapReduce components originally derived from Google's MapReduce and Google's File System [20]. HDFS provides the reliable distributed data storage and the MapReduce engine provides the framework to efficiently analyse this data [82].

2.4 The MapReduce Benchmark Suite (MRBS)

MRBS is a performance and dependability benchmark suite for MapReduce systems [64]. As shown in Figure 2.3, MRBS can emulate several types of workloads and inject different fault types into a MapReduce system. The workloads emulated by MRBS are designed to cover five application domains: recommendation systems, business intelligence (BI), bioinformatics, text processing and data mining. These workloads were selected to represent a range of loads, from the compute-intensive (e.g. recommendation systems) to the data-intensive (e.g. business intelligence - BI) workload. One of the strong suits of MRBS is to emulate client interactions (CIs), which may consist of one or more MapReduce jobs. These jobs are the examples of what may be a typical client interaction within a real deployment of a MapReduce system. MRBS and Hadoop constitute a "partly-open" [65] interactive system. New clients can arrive

at any time like in open systems. However, the system behaves as a closed system for the clients that are already connected. Namely, clients send a job to the JobTracker and then wait for the job to be finished, before submitting a new job. One job can contain multiple request.

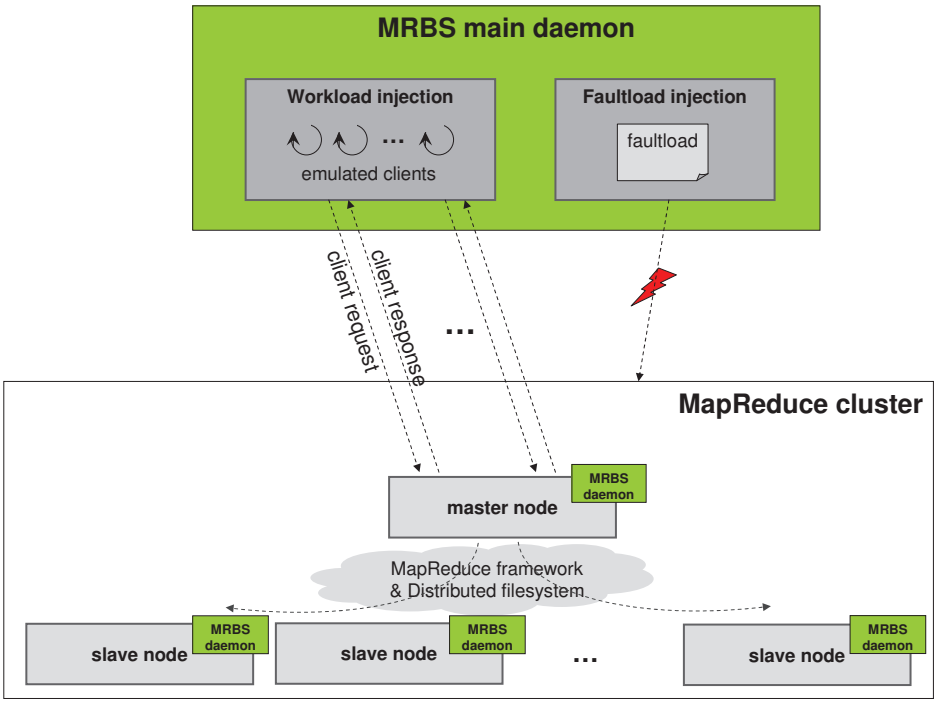


Figure 2.3: MRBS architecture

2.5 Definitions

Workload is defined as the number of clients (C) that are concurrently sending requests to JobTracker.

Cluster size (N) is defined as the number of processing nodes available for the framework.

Admission control is a classical technique to prevent server thrashing. In the MapReduce case it consists of limiting the maximum number of client request (MC), that are accepted by the central controller. This limit is put in place to ensure that all the accepted request have enough resources to run correctly.

Service performance is defined as the average time (y_{rt}) needed to process a request in certain time window. Low client response time is desirable as it reflects a reactive system.

$$y_{rt}[s] = avg(y_{rt_1}, y_{rt_2}, ..., y_{rt_N}) \tag{2.1}$$

y_{rt} is recalculated every 30 seconds, using a sliding window. For our experiments, the size of this window (T) is defined to be 15 minutes.

Service availability refers to the accessibility of the system to users per unit of time. MapReduce is available if the user requests are accepted at the time of their submission. Availability is instantaneous and concentrates on the fraction of time where the system is operational in the sense of being accessible to the end user. Availability (y_{av}) is measured as the ratio of accepted MapReduce client requests to the total number of requests, during a period of time.

$$y_{av}[\frac{\%}{T}] = \frac{N_{SuccessfulJobs}}{N_{SuccessfulJobs} + N_{RejectedJobs}} * 100 \tag{2.2}$$

T here is the previously defined sliding time window size, that is used to assign a measurable dynamics to the system.

2.6 MAPE-k loop vs Feedback control loop

Adaptation is considered an essential capability of many computing systems. Dynamic adaptation techniques have been studied by many different communities. In computer science one of the most used approach to dynamic adaptation is by means of a Monitor-Analyse-Plan-Execute-Knowledge (MAPE-k) adaptation loop.

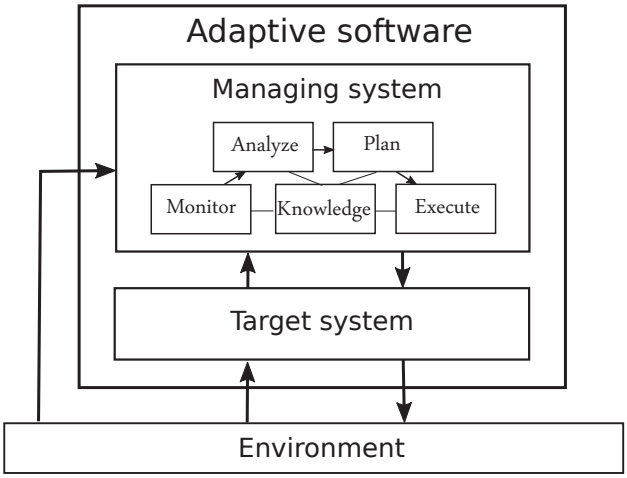


Figure 2.4: MAPE-k adaptation loop

Figure 2.4 shows the primary elements of an adaptive system following the MAPE-k structure. The *target system* is the system that we wish to control. It interacts with the *environment* and provides a service to the users. The *monitoring* component uses sensors to probe both the *target system* and its *environment*. This is done in order to build and maintain a model of the system in the *knowledge*. The *analyze* component decides whether adaptation is required

or not. If it is necessary, the *plan* component provides a plan with adaptation actions that the *execute* component will apply to the *target system* through the effectors.

Although this structure has been extensively studied in the literature, there are still several unsolved challenges. One such important question is how to provide assurances for software systems in uncertain environments, which is particularly challenging given the fact that systems have to be designed with partial knowledge and it is up to the runtime mechanisms to provide assurances. Applying classic techniques for providing such assurances (e.g. model checking) has proven difficult, therefore there is a clear need for new approaches to the engineering of adaptive software systems. One the most promising of these being control theory, which is a mature discipline that can provide a solid mathematical basis for synthesizing feedback control loops.

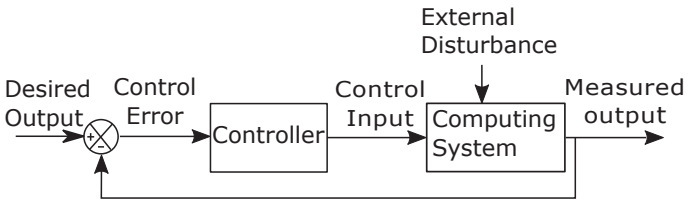


Figure 2.5: Feedback control loop

Figure 2.5 shows the primary elements of a feedback control system based on the principles of control theory. In control theory the adaptation process described in the MAPE-k loop case is called *control*, while adaptation refers to an adaptive controller. Namely, the feedback mechanism itself is subject to adaptation. If we compare Figure 2.4 and Figure 2.5, we can easily see that the target system in case of MAPE-k loop corresponds to the *Computing System*, while the managing system to the *Controller*. The model of the managed system in the MAPE-k loop is normally based on architectural concepts. In control-based adaptation on the other hand we usually have a mathematical model of the system (e.g. a differential equation). Furthermore, in feedback control, sensors are used to track the *Measured Output*, which is then compared to the *Desired Output*, resulting in the *Control Error*. The error is used by *Controller* to calculate the *Control Input*, that adapts the *Computing System* via actuators (also called knobs). The goal of a *Controller* is to minimise the *Control Error*.

2.7 Background on control theory for application to computing systems

2.7.1 White-box vs Black-box modeling

White-box models are based on the detailed modeling of the systems internal processes. However, in case of software systems, where there are no physical laws behind algorithms, such models are difficult to achieve and may become too complex for practical use.

In black-box modeling no assumptions are made regarding the internal structure and inner workings of the system. The model of the system is build by fitting a chosen model structure to experimental data. The general steps of building a black-box model are the following:

Step.1 Run exploratory experiments to form an initial idea about what the model structure should be. For example, one can see if the system responds linearly or non-linearly to input changes.

Step.2 Based on the experimental data, select the appropriate model structure and operating point.

Step.3 Design experiments to collect the dataset for the model fitting. During these experiments vary the inputs and measure the responses of the system. Care must be taken, that this changes uniformly cover all the input space, to avoid estimation bias. Moreover, as a general rule, the input signal should vary both in terms of amplitude and in terms of frequency.

Step.4 Collect the training data from the previous experiments, which consist of a set of input-output measurements.

Step.5 Run a parametric estimation algorithm to determine the model parameters.

For more information on black-box modelling please see [9], [41].

2.7.2 Discrete time signal

A signal is a concise way to describe a system characteristic that changes its value over time. In other words, a signal captures the evolution of a system variable over time. Signals can be deterministic or stochastic. For computing systems we can define several such signals of interest. For example response time, availability, throughput and cost.

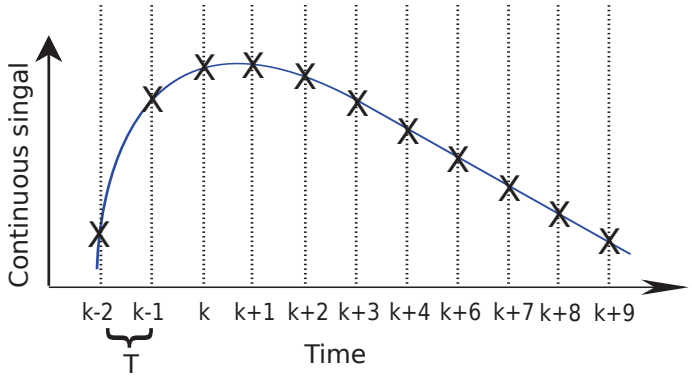


Figure 2.6: Fixed interval sampling of a continuous signal

In classical control theory we generally deal with deterministic discrete time signals defined as $y[k \cdot T] : \mathbb{Z} \to \mathbb{Z}$. By discrete, we mean that the signal is not measured at each time instant. Instead, as shown in Figure 2.6, the signal is sampled at every k^{th} multiple of a selected fixed sampling period (T_s). The end result of this sampling is a sequence of signal values ordered in time. There are several algorithms that provide formulas to calculate the size of T_s in order to accurately capture the information of the continuous signal [9].

2.7.3 Difference equations

While in case of continuous, physical systems the dynamics of the system is usually captured through differential equations, discrete systems are best modelled through difference equations. As there are no physical laws that govern how a software system is run, difference equations are generally preferable to differential ones.

Difference equations capture the effects of the system inputs on the outputs at discrete time intervals in time. Equation (2.3) presents a simple recursive difference equation that models, for example, the response time $y(k)$ dynamics of system. In this simple case the predicted response time depends on the current client count $u(k)$ and the previous response time $y(k-1)$, client count $u(k-1)$ values:

$$y(k) = a \cdot y(k-1) + b_1 \cdot u(k-1) + b_2 \cdot u(k) \tag{2.3}$$

where $k \in \mathbb{Z}$ here represents the index of the sample and $a, b_1, b_2 \in \mathbb{R}$ are called the model parameters.

2.7.4 Z transform

Z transform is a powerful mathematical tool to encode discrete time signals. It provides a compact way to write difference equations summing up the relation of the current signal value with past signal values. It is used to transform a discrete signal from the time domain into the frequency domain, where mathematical manipulation of the signals is simplified.

Instead of listing all the discrete signal values for each past k time instants, like given below

$$< y(k-\infty), y(k-1), y(k) >$$

we capture the information in a more compact form. Namely, as a sum of these previous values, using the following Z transform definition

$$y(z) = \sum_{0}^{\infty} z^{-k} y(k)$$

where $y(z)$ is called the Z transform of $y(k)$.

To get a better intuition behind the method, let us consider z^{-1} a delay operator, which denotes a delay of one sampling period of a signal. For example if we note the value of a signal at time instance k as $y(k)$, then $y(k-d) = z^{-d} \cdot y(k)$ denotes the signal value d sampling instances before.

Now let us consider the simple recursive difference equation given in Equation (2.3) that describes the dynamics of system response time $y(k)$. Using the delay operator z^{-1}, the Z transform of the model is:

$$y(z) = a \cdot z^{-1} \cdot y(z) + b_1 \cdot z^{-1} \cdot u(z) + b_2 \cdot u(z)$$

which is equivalent with

$$(1 - a \cdot z^{-1}) \cdot y(z) = (b_1 \cdot z^{-1} + b_2) \cdot u(z)$$

Furthermore, if we multiply by z everywhere we get

$$(z - a) \cdot y(z) = (b_1 + b_2 \cdot z) \cdot u(z)$$

and finally solving for $y(z)$

$$y(z) = \frac{b_2 \cdot z + b_1}{z - a} \cdot u(z) \tag{2.4}$$

Although the recursive form in Equation (2.3) and this latter Z transform formulations in Equation (2.4) are equivalent, the latter can be used to derive information concerning stability, settling times and especially for controller design.

2.7.5 Discrete time transfer function

In control theory a transfer function is a compact mathematical form of describing the effects of system inputs on its outputs. It captures how the output of the system is influenced by previous output and input signals. A discrete time transfer function is a mathematical representation of the system dynamics and is calculated by dividing the Z transforms of the output $Y(z)$ and the input signals $U(z)$:

$$G(z) = \frac{Y(z)}{U(z)}$$

If we take the previous example, given in Equation (2.4) and divide both side with $u(z)$ we arrive to the discrete time transfer function of the system presented in Equation (2.5):

$$\frac{y(z)}{u(z)} = \frac{b_2 \cdot z + b_1}{z - a} \tag{2.5}$$

which captures how the effects of client changes $u(z)$ affect the response time $y(z)$.

2.7.6 First order plus dead-time models - FOPDT

FOPDT models are fist order models that are commonly used to model systems with a delay in input actuation . The transfer function of such a model has the following structure:

$$G(z) = z^{-\tau}\frac{b}{(z+a)} \tag{2.6}$$

This delay τ means that the a change in the input signal can be observed on the output signal only after this actuation delay passes. In such a discrete form, τ is always defined as a multiple of the sampling period T. Using the properties of Z transform, Equation (2.6) can be transformed into the linear different equation form given in Equation (2.7):

$$y(k) = -a \cdot y(k-1) + b \cdot u(k-\tau) \tag{2.7}$$

2.7.7 Linear vs Non-linear Systems

All the systems of which output changes linearly in response to a linearly changing input and of which parameters do not change over time are called Linear Time Invariant (LTI) systems. Taking a more formal look at it, a system with transfer function G is called LTI if it satisfies the following two relations:

- The superposition principle: if we have two input signals u_1, u_2 and the transfer function G, the following relationship always holds.

$$G(a_1 \cdot u_1(k) + a_2 \cdot u_2(k)) = a_1 \cdot G(u_1(k)) + a_2 \cdot G(u_2(k))$$

- Time invariance: the output of the system does not explicitly depend on time.

For example, it easy to check that Equation (2.3) is a LTI system. Its output does not depend on time and its transfer function, given in Equation (2.4), satisfies the superposition principle.

To give an example of a time varying system let us modify Equation (2.3) into the following equation:

$$y(k) = a \cdot \boxed{k \cdot y(k-1)} + b_1 \cdot u(k-1) + b_2 \cdot u(k)$$

We can clearly see that in this case the output behaviour depends on the time instant k as well, therefore it is not time invariant.

Systems that have a non-linear response to a linearly changing input are called non-linear systems. As an example we can rewrite Equation (2.3) in order to have a non-linear (second order) relation between the client count and the response time:

$$y(k) = y(k-1) + \boxed{u(k-1)^2} + u(k)$$

We can see that although, the system is time invariant, it does not uphold the superposition principle.

2.7.8 System stability, performance and robustness

System stability is the most important property of any control system. Intuitively, a system is called stable if for any bounded input the output of the system also remains bounded. This definition implies that if our input is between certain limits, then in case of stable systems, our output will also be limited and not grow uncontrollably.

The advantage of the transfer function formulation, given in Equation (2.5), is that we can infer system stability, by simply calculating the roots of the denominator polynomial $z - a$. If the magnitude of all the roots is strictly less then 1, then the system is stable.

In control theory, when we speak of system performance, we think of how fast a system can follow a reference output trajectory or suppress the effect of any disturbances. While, robustness is the measure of how well the control system handles modeling uncertainties and changes in the system dynamics or in the environment.

Chapter 3

Related Work

Contents

3.1 Guaranteeing Performance in the Cloud 19
3.2 Mapreduce performance modelling 20
3.3 Controlling MapReduce Performance 22
3.4 Improving the MapReduce Framework 22
3.5 MapReduce Benchmarking and Simulation 24
3.6 Feedback Control of Computing Systems 24
3.7 Discussion 25

In this chapter we analyse the speciality literature to determine the current challenges in modelling and controlling the performance and dependability of cloud systems, taking a special look at our selected use case in the MapReduce framework. First in Section 3.1 we take a general look at the current issues regarding the performance control of cloud services. Then, a detailed overview of the state of the art methods for performance modeling, Section 3.2, and control, Section 3.3, of MapReduce systems is given. In Section 3.4 we emphasise that there is a large research effort in improving the performance and dependability of MapReduce systems by refining how the framework and its algorithms work. Through this chapter we desire to clearly differentiate our work from these, highlighting that in our case control means the change of only the configuration of the system and not its governing mechanisms. An other thing to note here although many solutions have been proposed to improve the dependability of a MapReduce framework, guaranteeing application level dependability of a MapReduce service has hardly been addressed in the speciality literature. Furthermore, we provide a brief overview of the existing MapReduce benchmarking and simulation approaches in Section 3.5. In addition, in Section 3.6 we take a general look at the state of feedback control usage in case of computing systems. Finally, Section 3.7 discusses the limitations of existing approaches and the open issues.

3.1 Guaranteeing Performance in the Cloud

The question of provisioning for performance guaranties has been addressed many times in the HPC, Grid and Database communities. Our work differentiates itself in several aspects. First

of all we present a novel method that enables the simplified, automatic modelling of complex computing systems. We chose MapReduce systems as our test case because it is a highly dynamic system in both data quantity, richness and in terms of its processing needs and it is one of the most popular current architectures for distributed data processing. Furthermore, we develop on-line control techniques that don't require complex tuning and that contrary to existing heuristic approaches, can provide mathematically proven guaranteed performance. Moreover, our approach is non-intrusive as it does not modify the framework. In addition, the developed techniques are sufficiently general to be applicable to a wide variety of cloud systems.

Current approaches to ensure performance in cloud systems can be separated into 4 categories: static, reactive, predictive and hybrid approaches.

i. Static deployments [75] are the standard in the industry and usually tuned based on the application peak demand and are generally make use of over-provisioning.

ii. Reactive approaches look at a certain output metrics, such as the current CPU utilisation, request rate, response time and add/remove servers as necessary [49, 4]. Reactive techniques are also offered by some public cloud providers, such as the Amazon Auto Scaler [6]. This provides the basic mechanisms for reactive controllers based on CPU usage measurements, but it is up to the user to define the static thresholds that trigger the scaling operations. Because of the simplicity of this mechanism there are several issues that arise in practice. First of all we think that for data intensive applications CPU is not a suitable metric for reconfiguration decisions. Moreover, we know that static shareholding is not optimal for each application and can lead to oscillations. More advanced reactive techniques can be observed at AutoScale [26], which uses queueing theoretical analytical models to decide capacity requirements based on the workload size and advocates for conservative downscaling to better manage server set-up costs.

iii. Predictive controllers that estimate the future load of a service based on past informations are also a popular approach. Several such algorithms have been proposed recently and are based on machine learning [47], Markov and fast Fourier transforms [28] or are using wavelets to provide predictions [50].

iv. Hybrid techniques combine the advantages of prediction and reaction [3]. Our work falls into the hybrid control category as we develop a control framework consisting of both reactive and predictive controllers.

3.2 Mapreduce performance modelling

Many studies have been already performed on how to model the performance the MapReduce framework. These can be grouped together into two large categories.

Analytical or first principle models are detailed MapReduce models that capture the inner workings of the different phases of a classical Hadoop MapReduce job execution flow, see [33,

40, 85]. Vianna et al. [78] propose a hierarchical model that combines a precedence graph with a queueing network to model the intra-job synchronisation constraints. Some as Jockey [24] use a simulator that captures the complex interdependencies of a job and makes use of previous runtime statistics to predict job runtime.

On the opposite side there are the regression and black-box models. These are coarse grained models that don't try to capture the specificities of the MapReduce framework but instead build upon job profiling, namely predicting the response time of future jobs based on past experience or exploratory runs. In the latter case the model parameters are generally found by running the job on smaller set of the input data and using regression techniques to identify model parameters. The differences between these regressive approaches lies mostly in the components used to set up the regressive model. Some authors develop statistical models made of several performance invariants such as the average, maximum and minimum run times of the different job cycles [77, 83].

Because as most MapReduce jobs are batch jobs that are run frequently, some propose building a profile database [77] to predict job runtime. This consists of a MYSQL database used to store past profiles of jobs and when the system has to execute a recurrent job the only thing needed is to search for the job profile in the database. They also further extend upon their initial invariants based model to include the effect of a single worker failure on the job performance and define upper and lower bounds for MapReduce jobs which can then be used for provisioning. Jin [36] suggests a stochastic performance model based on queuing theory to determine the best and worst case performances of MapReduce jobs in the presence of faults in the Map phase. Rizvandi [61] uses third order polynomials to predict the network load a MapReduce job. Others employ a static linear model that captures the relationship between job runtime, input data size and the resources allocated for the job [74].

Furthermore there are those who analyse long term traces to classify jobs into several runtime categories, for example from a 10-months logs of Yahoo's M45 supercomputing clusters running MapReduce. They use two separate algorithms for the prediction of service completion times: a distance-weighted average algorithm and a locally-weighted linear regression method. The linear regression based method proved to scale better for varying input sizes, see Kavulya [37]. Primary Component Analysis has been also used to determine the MapReduce/Hadoop components that have the biggest influence on performance of MapReduce jobs [84]. This approach mixes the non-application specific Hadoop configuration parameters with the statistical averages collected from the traces of previous job runs. They find, that as different applications have varying CPU, network bandwidth, data storage requirements, the use of clustering analysis is advised to group together jobs and build a separate model for every group to achieve better model performances.

To summarise, the existing models are job level models and therefore cannot capture the effects of workload variations in a MapReduce cluster. Furthermore these models are static and don't capture the dynamics of a MapReduce system. Namely, they capture only steady state of the system and not what happens during a workload change until the system reaches its new steady state.

3.3 Controlling MapReduce Performance

By controlling MapReduce performance we think of the on-line adaptation of frameworks resources or any of its parameters to achieve the required job deadlines.

SteamEngine [15] introduces an on-line performance and energy optimization algorithm for MapReduce applications running on virtualised clusters, such as Amazon EC2. It makes use of both off-line and on-line job profiling to predict the finish time of jobs. The performance optimization is done by regularly predicting the job finish time and using a simple heuristics to control the amount of resources available for tasks. Namely, if the predicted finish time, at any time during the jobs life-cycle is more than the expected finish time, then the algorithm increases the amount of resources (adds more nodes) through cluster scaling. The cluster scaling optimization is done only in the map phase, and the earlier it's done, the better the improvement will be.

Verma [76] proposes ARIA an automatic resource inference and allocation engine for MapReduce. ARIA can, at run time, allocate the appropriate resources (slots) to a job so that the jobs meets its time constraints.

Jockey [24] monitors job performance and dynamically adjusts its resources to maximise economic utility, while minimising its impact on the rest of the cluster.

While all the previous approaches propose fine grained job level performance control at a scheduler level, we propose to add course grained control by controlling the average performance of a group of jobs in the cluster. Moreover, while the existing techniques require modifying the schedulers and algorithms deployed by the MapReduce cluster, our control architecture is non-invasive and can be used in parallel with any of the previous listed algorithms. Furthermore, our algorithm can be easily automated to be used by an average user, without an in depth knowledge of the inner workings of the MapReduce framework. Meanwhile, the focus of fine grained scheduling techniques is to optimise the resources currently available, our course grained technique modifies the amount of resources, in order to handle workload spikes and fluctuations.

3.4 Improving the MapReduce Framework

There exist many attempts to improve upon MapReduce performance, either through framework modifications or by optimizing the framework parameters.

MROrchestrator [68] is an on-line resource management framework that can dynamically identify and resolve resource bottlenecks by using on demand reallocation of the distributed resources. The framework makes local estimates of the resource needs of different tasks, and through a central controller detects bottlenecks and controls resources accordingly. The controller uses a simple heuristic to control CPU and Memory allocations at node level. In their benchmarking application MROrchestrator leads to an average of 20% (max 38%) reduction of

job completion times and an average of 15% (max 25%) of increase in resource utilization. The authors compare their system to the two main contenders for improving resource scheduling and management, Mesos and Next Generation Mapreduce, and conclude that their approach has improved results, by more than 8%, in comparison.

Tarazu [2] proposes several additions to the MapReduce framework that affect the creation and scheduling of tasks. Namely, communication aware load balancing of the map stage, communication aware scheduling of remote map tasks and predictive load balancing in the reduce phase. They use 11 different benchmarks to test Tarazu, which achieves on average a speed-up of 40% over the original Hadoop implementation.

ADAPT [35] presents a novel data placement strategy for MapReduce. It builds upon the idea of distributing the data based on the availability of each node. This leads to improved data locality and reduced network congestion. For this they developed a stochastic model based on queuing theory to predict the performance of a task under interruptions. The algorithm outputs a hash table that is used to weigh the amount of data distributed to each node in the network. Experimental results show an average speed up of 30%.

Quiane-Ruiz [59] propose RAFTing MapReduce, namely improved fault recovery algorithms for MapReduce systems using check-pointing based techniques utilised in databases. They also implement a new scheduler that can take advantages of these check-points. Experimental results show RAFT outperforms Hadoop by an average 23% in the presence of task and node failures.

Hadoop's Ad-Hoc configuration also poses a great challenge for users and currently works mostly using some rules of thumbs given by Hadoop experts, still a large number of tuning parameters have to be chosen. Herodotu [34] introduces a dynamic algorithm to find the optimal Hadoop configuration for a given job. To achieve this, a profiler is proposed, that can collect statistical data automatically without modification to the MapReduce programs. Using the profile obtained, a simulator called What-if Engine is developed, that can simulate running jobs with different data, cluster sizes and Hadoop configurations. This fast simulator is called multiple times and uses subspace search techniques, such as Gridding (equispaced or random), Recursive Random Search, to automatically find the best Hadoop configuration to run the job in.

Sailfish [60] is a new MapReduce framework that, by aggregating the intermediate data, improves performance by batching disk I/O.

Hadoop++ [21] improves job performance for analytical queries using a new non-invasive indexing technique.

Recently, Facebook published a report in which is said that they reached the limits of the traditional MapReduce implementation. They identified several problems with the current implementations. One such problem was that at peak load performance cluster utilization drops due to scheduling overhead (max 70%). This is due to the fact that the heartbeat delay in scheduling is a problem in case of small jobs, when the slot based resource management

granularity is not small enough. An other issue found was, that software upgrades to the JobTracker implies whole cluster downtime. Facebook is not the only company suggesting that the traditional MapReduce implementations need improvement. Hadoop also proposes YARN, unofficially called MapReduce 2.0, as the new approach to solve the problems with the traditional implementation. In both of the approaches they separated the JobTracker's functionalities into two distinct entities: the Resource Manager and the Application Master. The Resource Manager is not job dependent and continuously optimizes cluster usage. A new Application Master is created for every new job and its tasks are job scheduling and monitoring. In addition, Facebook removed the heartbeat based monitoring model and introduced push based scheduling techniques. The advantages of the new system are better cluster utilization (95% for Facebook), lower job latency, no downtime necessary for upgrades, better resource management and scheduling fairness.

Furthermore, with the advent of cloud solution, there are many projects on improving MapReduce performance in the cloud. Spark [88] generalises the MapReduce model and can deal with new workload such as streaming, iterative algorithms and interactive queries. Although it is not yet as mature as Hadoop, it has been shown to outperform Hadoop by a couple of orders of magnitude in many cases. AsterixDB [12] is a new Big Data Management System that stores, indexes and manges semi-structured data. Because of its knowledge of data partitioning and indexing, it can avoid to always scan data to process queries. Stratosphere [11] further extends the MapReduce model, allowing for more operators then just map and reduce, and does much better on iterative algorithms then traditional Hadoop.

However none of these frameworks provide any control mechanism than can guarantee performance in face of a varying workload and uncertain environmental conditions. Moreover, due the generality of the algorithms developed in this thesis, they can be easily adapted to run with any of the previously listed frameworks to guarantee performance requirements.

3.5 MapReduce Benchmarking and Simulation

As there isn't a generally accepted framework to test and compare MapReduce behaviour with different cluster, input data sizes and fault conditions, we can find few benchmarking and simulation approaches in the literature. For example MRPerf [80] is phase-level simulator for MapReduce. In their test cases MRPerf is able to predict the performance of the map phase with an average 5.22% and for the reduce 12.83% error rate. MRBS [64] is a powerful benchmark suite, for the evaluation of dependability in MapReduce systems. It allows for injecting different fault schemes and contrary to previous popular benchmarking systems, such as HiBench, it supports concurrent job tests.

3.6 Feedback Control of Computing Systems

Many authors have argued for the potential of control-based adaptation of software systems. Hellerstein et al. [32] and Abdelhazer [1] advocated for applying control theory for computing

system adaptations more then a decade ago. They both highlight the promise of control theory in providing a mathematical basis for ensuring performance.

The increasing number of recent publications in the field of control of computing systems show the emergence of this new field for automatic control. For instance, continuous time control was used to control database servers [44] using Lyapunov theory and web service systems [55], HTTP servers [32], groupware servers [53] using a "black-box" approach. There have also been a few attempts to use continuous control as a method for dynamic resource provisioning of virtualised resources [52] or HDFS storage nodes [38].

Furthermore, several recent survey papers have emerged on the application of control theory to computing systems. Patikirikorala et al. provide a global view of the existing literature in this area [54]. Villegas et al. look at evaluation metrics for performance of software adaptations [79]. Guitart et al. conduct a survey on mechanisms for performance management of internet services [30]. Yfoulis et al. take an overview of resource provisioning approaches specific for the cloud [86]. In [25] a consortium of software engineers and control theorists underpin the major obstacles faced when applying control theory to software systems, such as finding mathematical models that can capture software dynamics in a form suitable for control synthesis and the lack of focus from software engineers on controllability when designing software systems. Finally, a growing interest is also emerging in the field of discrete event systems [63].

3.7 Discussion

From our analysis of the related work several open issues were identified.

- Current state of the art model based approaches to analyse and control the performance of cloud, respectively MapReduce systems, are based on static models. Meanwhile, the decades of experience in control theory in building control systems for physical systems has shown, that a model that can capture the dynamics of the system is crucial to decide when and how to control.

- Existing MapReduce models are fine grained job level models, hence they do not capture the effects of workload variability on the MapReduce cluster performance and dependability.

- Although lots of research is being done on improving how the MapReduce framework itself works still, none of the existing MapReduce cloud solutions give any guarantees in terms of performance and dependability. The reason for this is that, using classical software engineering techniques to provide assurances for systems designed with only a partial knowledge of the environment, is highly challenging.

- Poor end user experience, because of performance unpredictability, is still one of the major concerns with cloud service. Therefore, there is a clear need for application level,

multi objective controllers, that can be used as off the shelf add-ons to different cloud systems.

- More and more attention is given to dynamic adaptation techniques coming from other domains. The most prominent of these is control theory. Although dynamic adaptation using control theory has been gaining lots of attention in recent years [25], it has not been considered for use in adaptation of complex data parallel cloud service, such as MapReduce. As a result, there are also many unanswered questions from a control researchers perspective as well. How do we use the classical techniques to build a model of MapReduce systems? What do we measure? How do we chose our actuators? How do we manage the trade-off between performance, dependability and cost?

Chapter 4

Motivations and Objectives

Contents

4.1 Motivations 27
4.1.1 Impact of workload variations 28
4.1.2 Impact of cluster size variations 32
4.1.3 Impact of admission control 32
4.2 Objectives 32
4.3 Summary 33

In this chapter we first motivate our work by analysing current consumer concerns with cloud systems and examining the effects of workload variations on the performance and availability of a MapReduce cluster. The impact of possible corrective control knobs, such as cluster scaling and admission control, is presented as well. Finally, the objectives of the thesis are defined.

4.1 Motivations

A 2012 survey from Compuware [19] that asked 468 CIOs and senior IT professionals from the biggest companies in the Americas, Europe and Asia showed what are their biggest concerns regarding cloud service management. Their responses are summarised in Figure 4.1.

79% of them are concerned about the invisible costs of cloud computing. The biggest of these concerns (60%>) was the poor end user experience due to performance bottlenecks and heavy traffic, which in turn might lead to reduced consumer loyalty and loss in revenues. The reason for these concerns is that, while the scalability and provisioning mechanisms are there, the existing availability guarantees are in no means translatable to performance guarantees. Currently, the only guarantee given is that the servers are running and accessible. There are no guarantees in terms of application level performance and availability, which are seriously influenced by neighbouring applications, workload variability, and are the metrics that define the true end user experience. There is clear need for control mechanisms that can ensure these high level objectives, such as application response times and availability, and dispel consumer fears.

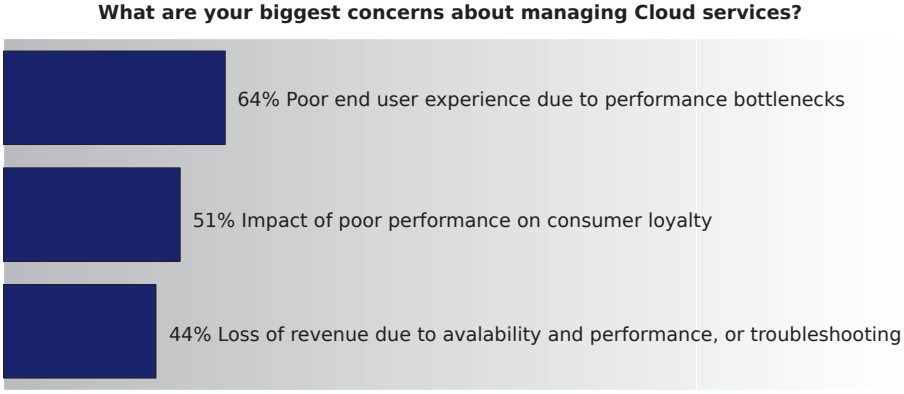


Figure 4.1: Cloud service management concerns by CIOs

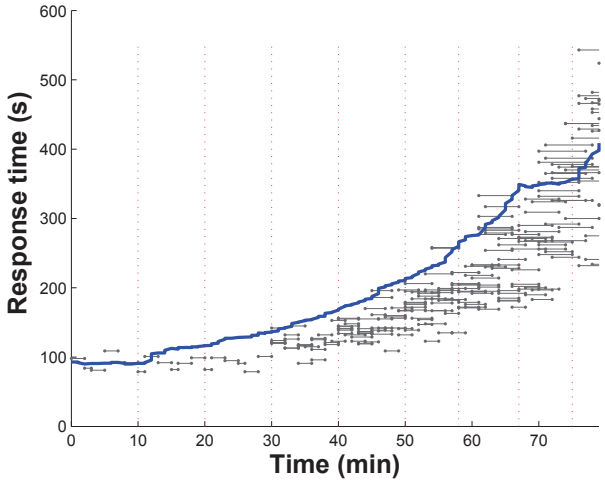
At the same time, as the MapReduce framework is being deployed more and more for time critical data processing applications, providing performance and dependability guarantees is becoming increasingly important as well. As it is the case with other cloud services, current MapReduce commercial solutions don't offer assurances either in terms of latency or availability. One of the reasoning behind this is that the most used MapReduce schedulers use fair scheduling policies, which were created to ensure high throughput and resource usage. Hence, while commercial MapReduce services, such as Amazon EMR [5] and Microsoft HDInsight [48], offer solutions for quick and cost-effective data processing, they don't provide any guarantees in terms of job runtimes. Although elasticity mechanisms are given, they are not completely automatic and it is up to the user to define the scale up/down thresholds. Therefore several not trivial questions are left up to the service user, such as:

- How many resources does my application initially need?
- How to handle the effects of workload variations on performance and dependability?
- How to ensure certain response times and continuous service availability ?

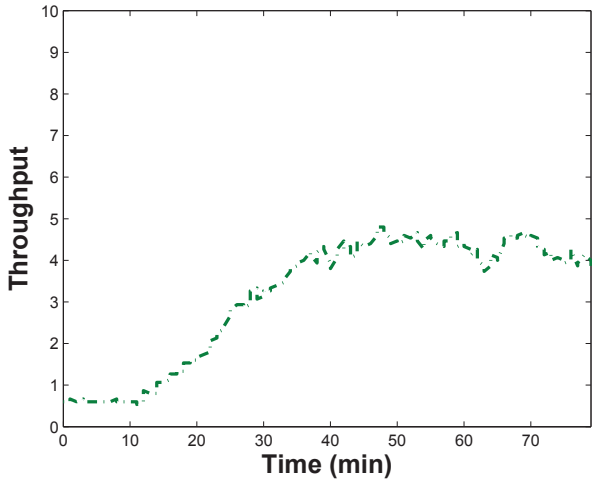
To be able to answer such questions, first we take a look at what are the effects of workload variations, cluster scaling and admission control on the performance, throughput and availability of MapReduce systems.

4.1.1 Impact of workload variations

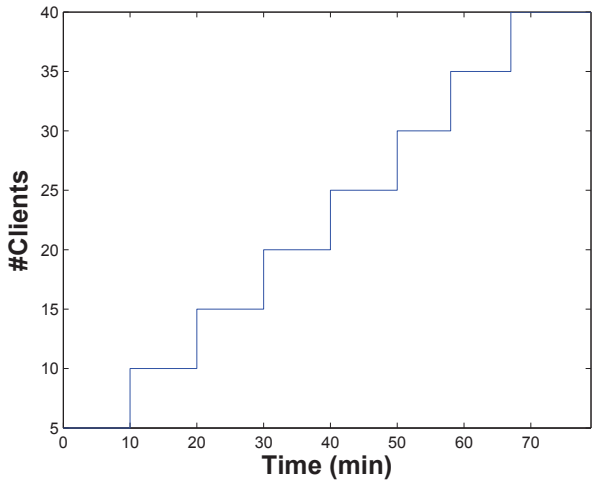
As defined in Section 2.5 workload is defined as the number of clients (C) that are concurrently sending requests to the MapReduce service. Figure 4.2 shows what happens when this workload is varied, while keeping all other metrics unchanged. The cluster size is fixed to 20 and the number of clients is increased gradually from 5 clients to 40. Every 10 minutes 5 clients are added (Figure 4.2c). The jobs during the experiment are represented by multiple horizontal lines. The beginning of a line is where a job starts and the end of each line is when the job ends. The jobs are grouped together vertically based on their runtime and chronologically on the horizontal axis.



(a) Performance

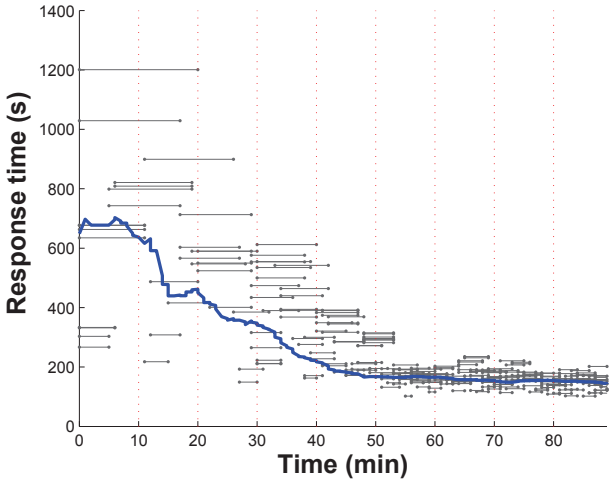


(b) Throughput

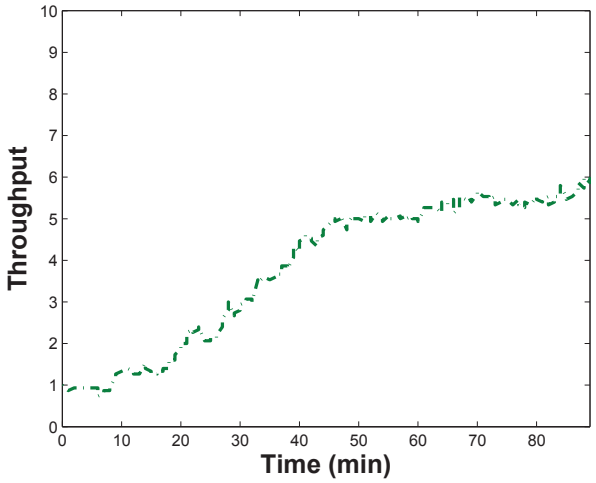


(c) Clients

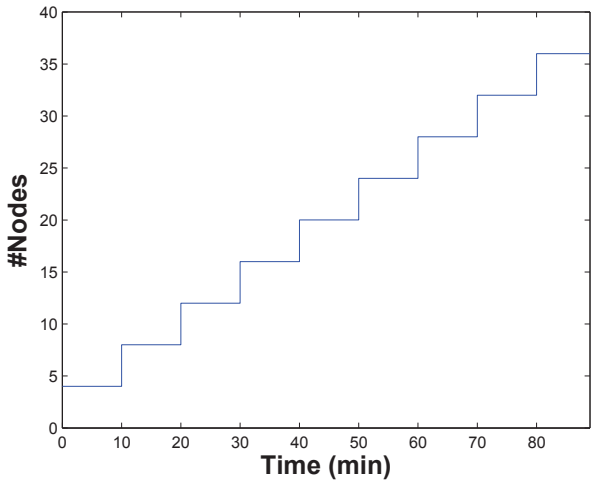
Figure 4.2: Effects of workload variation, #Nodes=20



(a) Performance

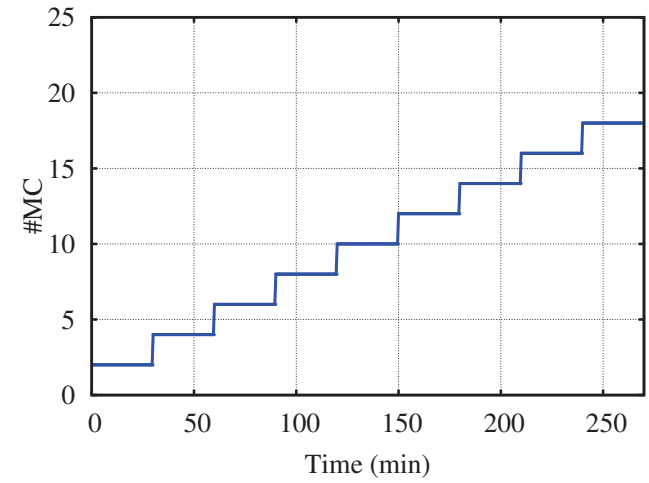


(b) Throughput

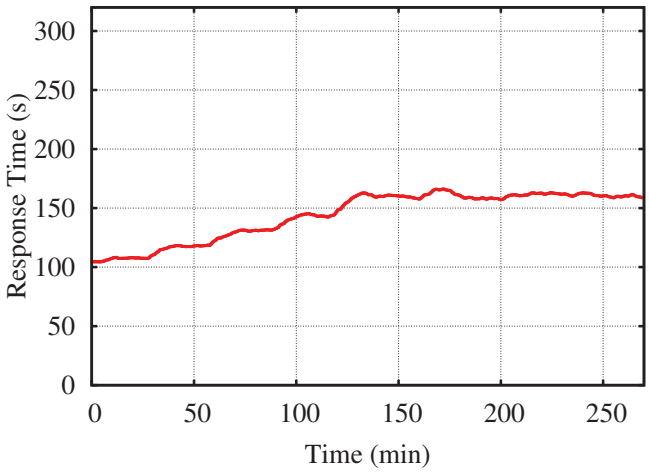


(c) Cluster size

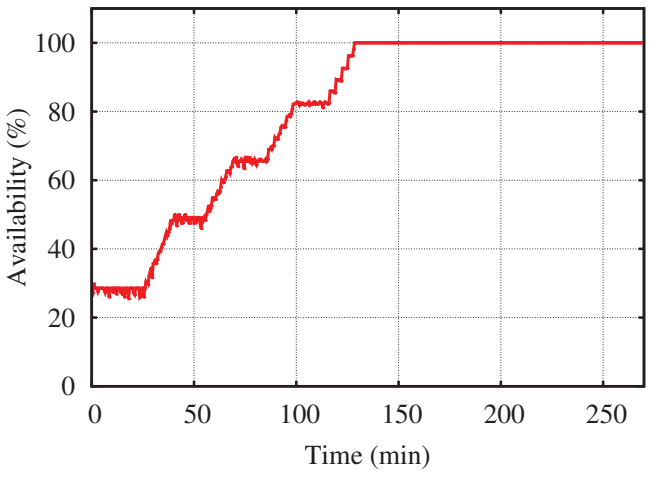
Figure 4.3: Effects of cluster size variation, #Clients=10



(a) MC



(b) Performance



(c) Availability

Figure 4.4: Effects of admission control level variation, #Nodes=20, #Clients=10

If we take a look at Figure 4.2a and Figure 4.2c we can see that as the clients are increasing linearly, the average service is increasing exponentially. Figure 4.2b shows that the throughput increases linearly until it gets saturated, after all the cluster resources have been occupied.

4.1.2 Impact of cluster size variations

The experiment in Figure 4.3 presents the results when the cluster size is varied from 4 to 36. The number of concurrent clients is fixed to 10 during the whole experiment. 4 new nodes are added every 10 minutes (Figure 4.3b). As in the previous figure, the multiple horizontal lines in Figure 4.3a constitute the runtime of the different client interactions. It can be seen that the overall behaviour of MapReduce is non-linear, since the proportional increase in the number of nodes is not proportional to the reduction in response time and the increase in throughput. It is also interesting to notice that, although the throughput gets saturated in this case as well (Figure 4.3b), it is not for the same reasons as in the previous case, when it was because of lack of resources. In contrast, here the throughput is saturated when we have more resources then required by the current workload.

4.1.3 Impact of admission control

For implementing admission control a proxy based approach is taken. The proxy allows no more then a selected $\#MC$ number of concurrent requests to the JobTracker. Any client request above this level is rejected, which maintains performance, but in turn negatively affects availability. If we take a look at Figure 4.4 for example, we can see that as the MC level is increasing, more and more clients are allowed to connect to the cluster and both the response time and availability are increasing, until the MC level reaches the number of concurrent clients C. After this point, MC level has no influence on our output metric of performance and availability, since we are allowing for more requests then needed at the time.

4.2 Objectives

From the previous analysis of the impact of different metrics on performance and availability of a MapReduce service, we can clearly see that both cluster size and admission control have a strong impact on the quality of service of MapReduce systems. In the case of admission control the trade-off between performance and availability must be taken into consideration. Meanwhile, a similar compromise must be made between cost (cluster size) and performance. Workload variations are uncontrollable and can be considered a disturbance to the system, having adverse effects on both performance and availability.

Taking all this into consideration, in the following we define the objectives of the thesis. We choose as tunable control parameters the cluster size (N) and the max clients level (MC), while the number of clients will be treated as a measurable disturbance to the system.

Our first task is to build a test-bed where one can easily develop and test multiple different modeling and control strategies of a real MapReduce service. To facilitate further research by control scientists into such computing systems, a control interface is required between Matlab, the standard development environment in automatics, and the MapReduce cluster. In the following, a short list of these *technical objectives* is presented:

- Construct the actuators to control on-line the cluster size and the max clients level of a MapReduce cluster.
- Build sensors that can measure on-line output metrics, such as response time, availability and workload size.
- Devise an interface between a local Matlab controller and the remote sensors and actuators of the MapReduce cluster.
- Create an autonomous control framework that can control on-line the cluster performance and availability, from any local computer running Matlab.

For a detailed view of our experimental setup that implements these objectives take a look at Section 7.1.

Furthermore, taking into account the open issues found during the overview of the speciality literature and highlighted in Section 3.7, the *scientific objectives* of the thesis are defined as the following:

- Design a model that can capture the dynamic performance and availability of a MapReduce cluster, running a data intensive concurrent workload. The dynamic model must be compatible with control theoretic techniques (Chapter 5).
- Design control laws capable of guaranteeing quality of service objectives for a MapReduce application in term of performance, availability and cost (Chapter 6).
- Evaluate the proposed models and controllers experimentally on a real MapReduce cluster (Chapter 7).

4.3 Summary

In this chapter we further motivate the need for on-line application level control of the performance and availability of cloud services. Furthermore, the negative effects of workload variability are highlighted with concrete experimental results. We show that control knobs, such as cluster scaling and admission control, are suitable for the performance and dependability management of a MapReduce system. Finally, the technical and scientific objectives of the thesis are defined.

Chapter 5

MapReduce Performance and Availability Modelling

Contents

5.1 Capturing dynamic behaviour of software systems 36
5.1.1 Challenges 36
5.1.2 Assigning measurable dynamics to software 36
5.2 Building the MapReduce dynamic model 37
5.2.1 Assumptions 37
5.2.2 Model structure 38
5.2.3 Analysing MapReduce dynamic behaviour 39
5.3 MapReduce Performance model 39
5.4 MapReduce Availability model 46
5.5 MapReduce Performance and Dependability model 47
5.6 Model identification 50
5.6.1 Off-line identification of the model parameters 50
5.6.2 On-line adaptation of the model parameters 52
5.7 Summary 53

As described in the related work section, multiple models for MapReduce systems exist in the speciality literature, however most of them focus on capturing the steady state behaviour of cloud systems. Meanwhile, the decades of experience in building control systems for physical systems has shown that capturing the dynamics of the system is crucial, to determine when and how to efficiently control. Therefore, our first objective is to find a dynamic model of a MapReduce system. First, in Section 5.1 we introduce our chosen methodology for assigning a measurable dynamics to software systems and we perform a preliminary analysis of MapReduce dynamics. Second, the general MIMO model structure for MapReduce systems is presented in Section 5.2.2 and then we gradually identify the correlations between each input and the outputs. A performance model, that captures the connection between the cluster, workload sizes and the average runtime is presented in Section 5.3. The relation between availability and the max clients level is developed in Section 5.4. The two previous models are combined and the general MIMO model of the MapReduce system is described in detail in Section 5.5. Finally, the off-line and on-line parametric identification methodologies are presented in Section 5.6.

5.1 Capturing dynamic behaviour of software systems

5.1.1 Challenges

Although control theory has been already applied for modelling of computing systems since the beginning of the 2000's, we believe that there is still large reticence from both control theoretical and computing communities in applying such techniques in practice. Mathematical modelling of cloud software systems poses several specific challenges. These difficulties range from the simpler ones, such as language difficulties (for example the notion of 'control' means something different for both communities), to more complex ones, such as the difficulties in modelling software applications due to lack of physics behind algorithms and services.

Still, finding models that capture software dynamics with sufficient accuracy is at the same time critical and unusual for both communities. From a control perspective the challenges are the following:

- One of the most critical questions for software systems is the choice of Input/Outputs. While most physical system are build to be controllable, most computing systems are not designed with on-line control in mind. Therefore, the choice of tuning parameters and performance measures is not straightforward any-more. Furthermore, contrary to physical systems there are numerous possible tuning variables and as a results it is difficult to decide between them.
- Building models is unusual. Because there is no physics behind applications finding a first principle model, based on traditional laws of physics, is very difficult. Nonetheless, for most computing systems we have an in depth knowledge of how the system works, therefore, in theory one could build an almost perfect mathematical model of the system. However, in most cases such a precise model would be too complex for practical usage.
- During their life-cycle, software applications are always changing due to frequent updates, thus evolve rapidly. Therefore, in order for any model to be relevant, special care has to be taken to remain implementation agnostic and include only high level metrics that will not disappear during future updates.

5.1.2 Assigning measurable dynamics to software

Our control objective is selected as keeping the average response time below a given threshold, while keeping availability above a predefined limit and costs to the minimum. To achieve this, one has to assign a measurable dynamics, that is to say the construction of a discrete-time signal as it is presented in Section 2.7.2, to the output metrics.

The key point of a signal is that it considers the evolution of the system over time. Finding such a signal could be done by defining a fixed sampling time T_s and measure the value of response time or availability at periodic instants. The issue with such an approach is that

we should have well defined, measurable values of these metrics at exactly those periodic sampling instants, which is not the case in practice. A more practical approach is to calculate a statistical aggregate over a fixed time period in the past at each Ts time instant. This technique is called a sliding window over time.

The chosen aggregate function can be anything from a *mean*, *median* (50^{th} percentile) to the 98^{th} *percentile*. The choice of this function is not straightforward. For example, the *mean* function is found to be highly sensitive to the presence of outliers in the data, while in comparison percentile measurements are very robust to this. However, when we don't have outliers in the data the *mean* value usually provides a more accurate measure of the central tendency. *Mean* measurements, can also be utilised in case of skewed data, but only with additional pre-processing of the measurement data that eliminates any outliers, for example those that fall outside a given standard deviation value. Nevertheless, in this case caution has to be taken, that if measurements do not follow a bell curve, standard deviation calculations can result in erroneous tendency approximations. To summarise, *percentile* measurements are more suitable when we have a data set that has a skewed, long tailed distribution. Moreover, it can provide the advantage of focusing on certain desired signal areas. For example, a controller that wants to improve the general performance can focus on $90^{th}\%$ measurements, while one that has as objectives the elimination of outliers can use $98^{th}\%$ measurements.

One other challenging question when working with time windows is choosing the size of the window. The bigger the window is, the more we loose system dynamics, while the smaller it is, the larger the noise will be in the measurements. In our case the window size is tuned off-line. To choose the windows size we start with a small value and increase it gradually until we reach the desired signal variance and the curves smoothen out. Below this size the output measurement may be influenced by the noise that arises from the natural variance of the jobs. From a control perspective, if the window is bigger than this size, then the controller reacts slower and if it is smaller it will react to noise.

5.2 Building the MapReduce dynamic model

5.2.1 Assumptions

Before we start the modelling we define our key assumptions about the MapReduce system in question:

- Our system is a "partly-open" [65] interactive system. Namely, new users can arrive at any time like in the case of open systems. However, the system behaves as a closed system for the clients that are already connected. This means that the clients send a job to the JobTracker and then wait for the job to be finished, before submitting a new request.
- Although in the general, the jobs runtime in case of MapReduce systems can range from the minutes to the hours magnitude, we assume that the variance in the mean

response is smaller then 25%. This assumption is necessary because of our use of mean as measurement aggregator. In practice, this can be done through using simple clustering algorithms based on past history job run-times. Moreover, this assumption is not necessary if one uses a percentile based aggregator. The advantages/disadvantages of both methods have been discussed in detail in Section 5.1.

- No failures are present at the time of modeling. Although MapReduce runs on top of cheap commodity computers in general and can handle multiple types of failures which impact system performance, we are interested in modeling the failure free dynamic behaviour MapReduce. Even though we don't explicitly model these kinds of failures, our control architecture can correct their effects on the system performance, failures being treated as unmodeled disturbances.

- We assume that the workload buffer is not empty at steady state. If all the clients leave, new dynamics might be observable while the buffer fills up or empties and the model is no longer valid.

5.2.2 Model structure

The choice of control inputs out of Hadoop's many parameters (more than 170) is also not straightforward. As we set out for our model to be implementation agnostic, we take into consideration only those parameters that have a high influence regardless of the MapReduce version used. Two such factors, that have been identified having among the highest influence, are the number of Mappers and the number of Reducers available to the system [84]. Since the number of Mappers and Reducers are fixed per node level in MapReduce systems, we chose the *cluster size* (N) as our first control input. Our second control input is the *max clients level* (MC) which is a tunable parameter that controls the number of concurrent requests sent to the JobTracker. The number of concurrent *client* requests is treated as an uncontrollable (exogenous) input.

MapReduce *performance and availability* are chosen as our output metrics. By performance we think of response time, defined as the average time (y_{rt}) needed to process a request in certain time window. While availability is measured as the ratio of accepted MapReduce client requests to the total number of requests. See Section 2.5 for the formal definition of these metrics.

Based on these observations, the following model that captures the dynamics of MapReduce systems in terms of performance and availability is proposed. The general structure of our model can be seen in Figure. 5.1.

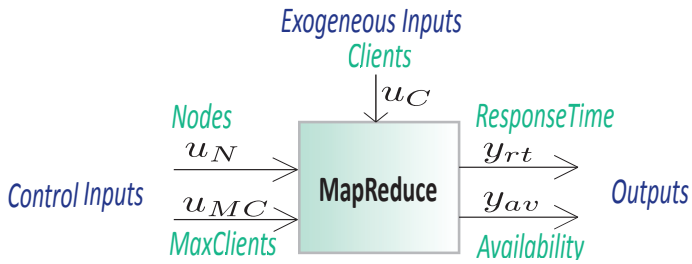


Figure 5.1: System model with specified inputs/outputs

5.2.3 Analysing MapReduce dynamic behaviour

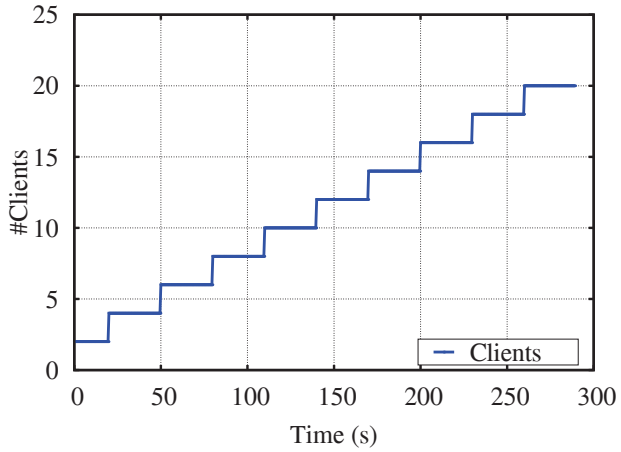
Before deciding on a model structure, exploratory experiments need to be run to see the effects of the different inputs on the outputs. This kind of exploratory experiments allow a control scientist to decide upon the order of the polynomials to use when defining the model structure.

In the first experiment shown in Figure 5.2, we analyse the performance and availability of the MapReduce system in case of a varying workload. The cluster size is fixed to 20 nodes and the max clients level is set to 10. As seen in Figure 5.2a the workload size is varied by adding 2 clients every 30 minutes from 2 clients to 20. The second experiment in Figure 5.3 presents the results when the cluster size is varied, by increasing the number of nodes from 5 to 20, adding 5 more nodes at every 30 minutes. Here, the workload size is set to 10 and the max clients level is fixed to 5 during the whole experiment. The third experiment presents the effect of max client level variations on performance and dependability of a MapReduce cluster, meanwhile fixing the cluster size to 20 and workload size to 10.

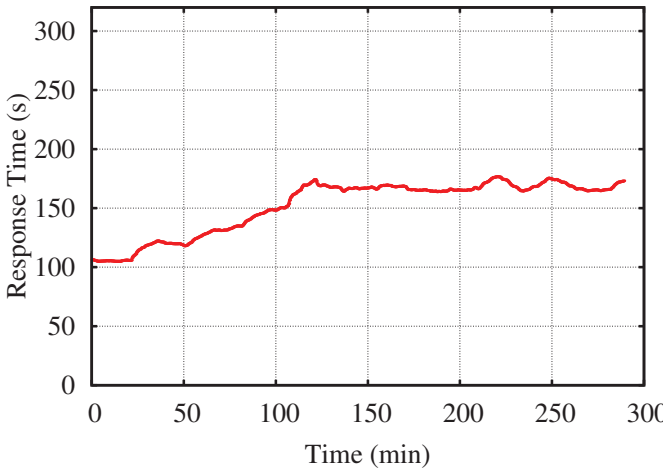
If we analyse the system behaviour depicted in Figures 5.2 to 5.4, it is clear to see that the system is nonlinear in the considered range of input variations. Furthermore, a quick look at Figure 5.3c combined with the analysis of Figures 5.2c and 5.4c, makes us notice that the availability y_{av} is influenced only by the number of clients C and the max clients level MC, but not by the number of nodes N that the system has. Moreover, we can see that the control input MC has no effect on the availability if $MC > C$. The system becomes linear only if $C > MC$.
In the following sections we detail how the relation of each output with our inputs is identified.

5.3 MapReduce Performance model

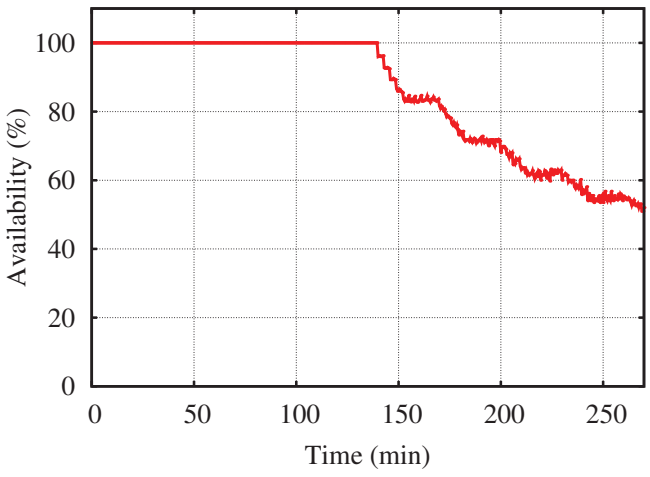
In order to build a performance model for a MapReduce system, first we have to analyse the correlations between our performance metric response time and the system inputs, cluster and workload sizes. Concerning the response time (y_{rt} - the first output), we had already seen in the exploratory experiments presented in Figure 4.3 (p. 30) in the motivations section, that the system is non-linear with respect to cluster size changes.



(a) Clients

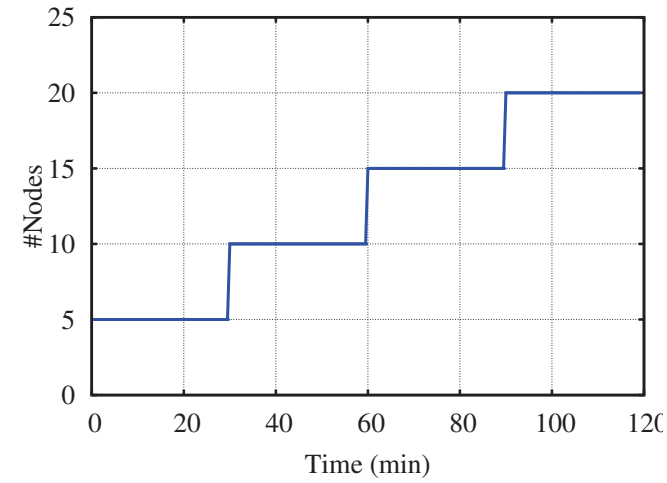


(b) Performance

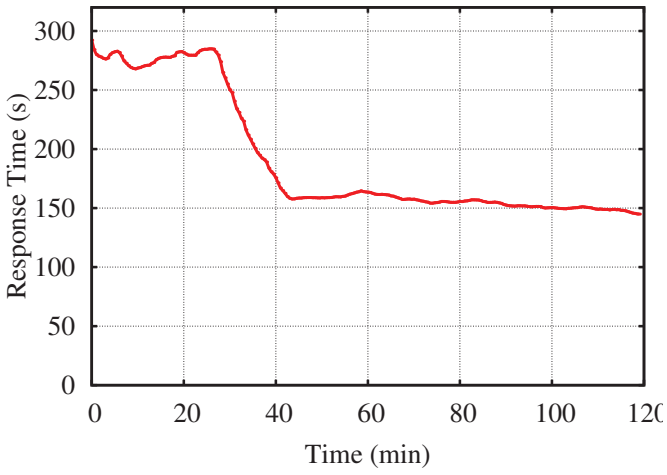


(c) Availability

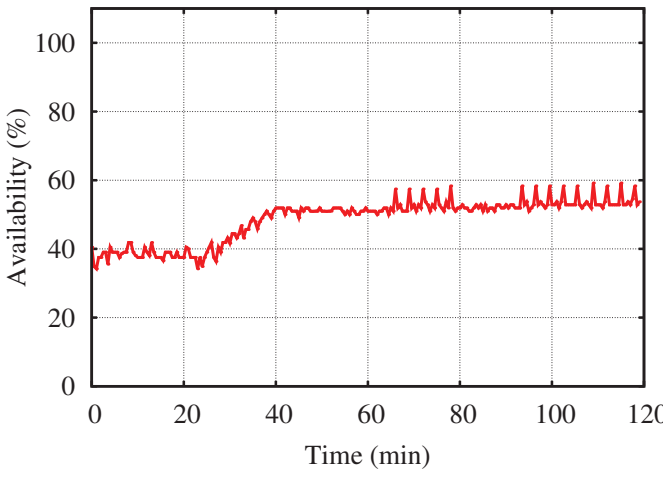
Figure 5.2: Effects of workload variation, #MC=10, #Nodes=20



(a) Cluster size

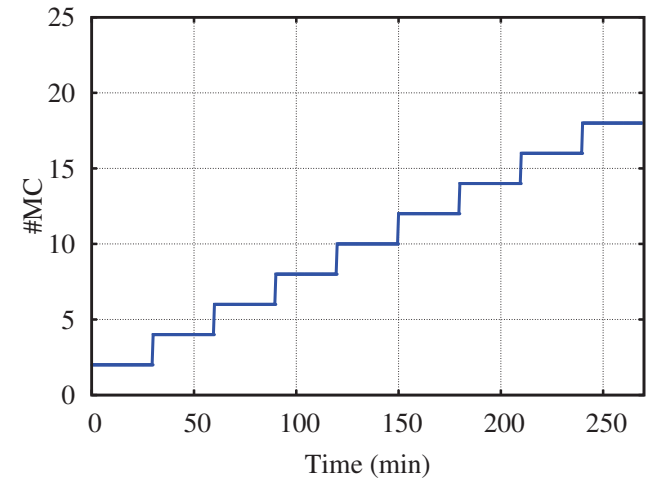


(b) Performance

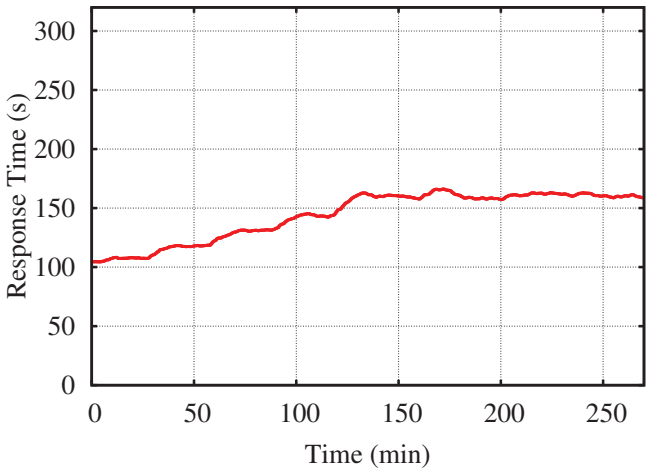


(c) Availability

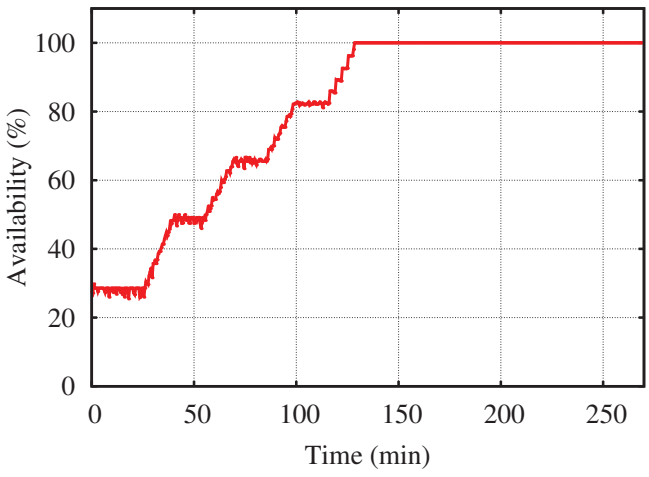
Figure 5.3: Effects of cluster size variation, #MC=5, #Clients=10



(a) MC



(b) Performance



(c) Availability

Figure 5.4: Effects of MC variation,#Nodes=20, #Clients=10

While, when it comes to the workload size changes in Figure 4.2 (p. 29), we observe three specific response time behaviours:

1. Initially the system resources are underutilised and therefore every request gets all the required resources for optimal performance.
2. The second phase starts when all the free resources have been occupied and the clients start sharing the existing ones. Starting from this point, response time starts increasing linearly as the number of clients increase.
3. When too many clients are added we enter the third phase, when the response time will start to increase exponentially as more clients arrive.

If we desire to model the dynamics of such a non-linear system, a control theoretician has two possibilities. Either try and capture all the system non-linearities into a general, but complex model or linearise around a desired operating point and get a less general but more practical model. Linearisation is generally preferred when possible, because if we have a linear model, one can take advantage of all the well know classical techniques for control design, developed over these past decades. Taking the latter reasoning into consideration we suggest to build a linear model, that is a linear approximation of the non-linear model in the desired operating region. The natural question that arises is, how do we find this region in case of a cloud service.

A typical system design time question that might arise is that, given an expected number of clients, what is the minimal amount of resources the system needs, so that all the requests have their necessary resources and therefore run as fast as possible? Based on our experiences with modelling MapReduce systems, we propose the following general algorithm, that answers the previous question, and which can be used to find the linearisation point for cloud software systems in general:

1. Select the initially expected average workload amount to be served based on financial constraints.
2. Increase the cluster size until system throughput starts to saturate.
3. Set this point of saturation defined by $(\#resources, \#clients)$ as your set point for linearisation. Here the system is fully utilised. Adding more clients would decrease performance, adding more resources would not improve performance.

To better understand the reasoning behind the previous algorithm, let us perform a simple thought experiments. First, let us suppose we select the initial average workload amount we desire to serve, based on financial constraints, at 10 clients for example. Now we start increasing the number of nodes until the point when the throughput starts to saturate (at 20 nodes for example). At this point we have the exact number of nodes with which we can optimally serve the current amount of clients, while having maximum utilisation of the

resources. Adding more resources would not improve upon performance, it would just lower utilization. In the following we analyse what are the advantages/disadvantages of choosing the linearisation point in this manner.

Advantages:

- The main advantage is that we will have a model of our system at the edge of full utilisation, which is where we normally want our steady state of the system to be, to minimise costs.
- The model can predict what happens when more clients arrive at full utilisation and can help a controller decide upon the number of resources to add to reach once again full utilisation and keep the desired performance levels.
- These type of models are intuitively simple and therefore easy to understand for system engineers without a control theoretical background.

Disadvantages:

- The model is valid only around the operating point. As a consequence, if we model our system using 10 clients and 20 nodes and then test our system with 100 clients and 200 nodes the model might loose validity, depending on system linearity.

  However, this disadvantage can be easily addressed either by re-triggering the identification procedure when we leave the operating region and/or by on-line adaptation of our model parameters. For more details concerning the identification procedures see Section 5.6.

Taking all this into consideration the structure of proposed dynamic model that predicts MapReduce cluster performance based on the cluster and workload sizes can be seen in Figure 5.5. Our control input $N(k)$ is the cluster size, while the changes in clients $C(k)$ is considered as a measurable disturbance. Our output $y_{rt}(k)$ is the average response time of a job in the k^{th} time interval.

We recall here that our objective is to mathematically quantify the effects of cluster size and workload changes on our output (response time). Since the sum, shown in Figure 5.5, is linear around the linearisation point previously computed, we identify them separately in order to get the mathematical relations between each input and the output. Through the sum of their effects we get a multiple input, single output (MISO) model. This technique is based on the superposition property of linear time invariant systems, defined in more details in Section 2.7.7, which allows for the identification of the input effects independently from each other. This is how we arrive to Equation (5.1).

$$Y_{rt}(z) = Y_C(z) \cdot C + Y_N(z) \cdot N \tag{5.1}$$

where $Y_N(z)$ and $Y_C(z)$ are discrete time transfer functions. A transfer function is a compact form of describing the effects of system inputs on its outputs, see Section 2.7.5 for more

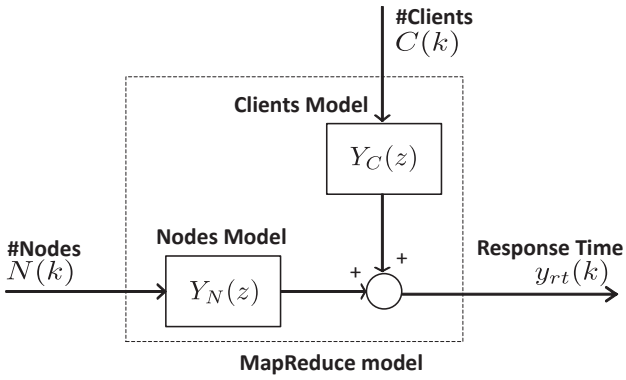


Figure 5.5: MapReduce performance model

details. In our case $Y_N(z)$ and $Y_C(z)$ capture respectively the effect of cluster size variation on response time and the relation between response time and the workload size. These functions can have any structure at this point. The methodology for choosing their structure is given in Section 5.6.

The observed linearity in the operating regions, the lack of overshoot and exponential decay all indicate that the response could be modelled, at least in a first approach, with a first-order linear difference model with deadtime (FOPDT). An introductions to FOPDT models can be found in Section 2.7.6. Moreover, these kind of models are widely used in the control community for a wide variety of systems and have been proved to provide a sufficiently good system fit even if the real system dynamics has a higher order dynamics [29]. Substituting Y_N and Y_C transfer functions in Equation (5.1), with a FOPDT structure we get Equation (5.2), presented below:

$$Y_{rt}(z) = z^{-\tau_{rt_c}} \frac{b_c}{(z + a_c)} \cdot C + z^{-\tau_{rt_n}} \frac{b_n}{(z + a_n)} \cdot N \tag{5.2}$$

In this equation the coefficients a_c, b_c, a_n and b_n along with the potential delays (τ_{rt_c} and τ_{rt_n}) are to be found by the identification algorithm detailed below in Section 5.6. One crucial part of these models are the actuation delays which are an important, and a somewhat overlooked issue in the speciality literature, when designing controllers for the clouds.

These delays can have multiple sources in cloud scenarios, ranging from boot up times (process, node) to delays caused by the data processing processes. A good example for these is when we add a new node to the MapReduce cluster. Its effect on the response time is not instantaneous, as new jobs have to finish that made use of the new resources, before we can measure the effect of the node on our output. In addition, the JobTracker has to copy/replicate the data to the new nodes before assigning it tasks. In our opinion, it is highly important to consider these delays in our models, otherwise the controller will keep on adding unnecessary nodes until it can see their effect on the measurements which will lead to an oscillating control profile.

Moreover, the model presented in Equation (5.2) can be further generalised by not fixing the model order to a be first order transfer function. To provide such a model, we transform Equation (5.2) using the Z transform properties and propose the following linear difference equation as a general performance model:

$$y_{rt}(k) = -\sum_{i} a_i \cdot y_{rt}(k-i) + \sum_{j} b_j \cdot C(k - \tau_{rt_c} - j) + \sum_{l} c_l \cdot N(k - \tau_{rt_n} - l) \tag{5.3}$$

where model parameters such as:

- a_i capture the effect of past outputs on current outputs
- b_j capture the correlation between current outputs and past workload sizes
- c_l capture the correlation between current outputs and past cluster sizes

Equation (5.3) is given in difference equation form, as it is simpler to understand than the transfer function formulation shown in Equation (5.2), for people without a control theoretical background. Therefore this formulation is preferable whenever we have multiple communities working together.

What Equation (5.3) captures, is that at any time instant k the dynamics of response time is defined by a weighted sum of past response times, cluster and workload sizes. The delays in the actuation are represented by τ_{rt_c} for workload size C and τ_{rt_n} for cluster size N. These delays in discrete form are always calculated as a multiplicative of the sampling period T. In the form presented above, if $\tau_{rt_c} = 5$ for example, then $C(k - \tau_{rt_c})$ refers to the cluster size 5 sampling periods ago.

5.4 MapReduce Availability model

The structure of proposed dynamic model that predicts MapReduce cluster availability based on the max clients level $MC(k)$ and workload size $C(k)$ can be seen in Figure 5.6. Our output $y_{rt}(k)$ is the average availability of a job in the k^{th} time interval. As in the case of service availability, the effect of clients rejections are almost instantaneous on the availability value and therefore, in this case, we don't need to consider delays in the actuation. Consequently, we suggest a first order relation between *availability* (y_{Av}) and our inputs clients (C) and max clients (MC) in form of the following input discrete time transfer function:

$$Y_{av}(z) = Y_{MC}(z) \cdot sat_0^{\infty}(C - MC) \tag{5.4}$$

where the saturation is defined as:

$$sat_a^b(x) = \begin{cases} b & \text{if } x > b \\ x & \text{if } x \in a, b \\ a & \text{if } x <= a \end{cases}$$

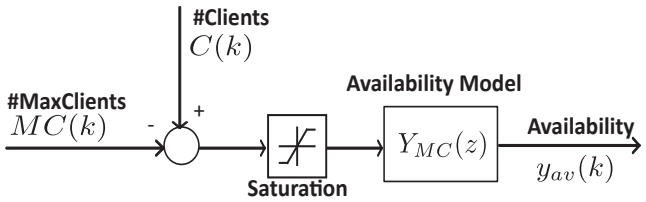


Figure 5.6: MapReduce availability model

As we can see, although $Y_{MC}(z)$ is linear in this case, we do have an input non-linearity to consider, namely input saturation. The intuition behind this saturation, utilised in Equation (5.4), is the following. The dynamics of availability depends upon the number of clients C that are above the max clients MC limit. When the client number C is below MC, availability is not influenced by the workload size. Hence, when $(C - MC)$ the difference is negative the availability remains unchanged.

Although it is important to consider saturations when modeling, in general saturations are difficult to treat when designing our control laws. Therefore, we remove the saturation from our model by making the following control design choice:

The controller controls the MC level in such that: $\forall k \in \mathbb{Z} | C \geqslant MC > 0$

Taking this into account our availability model, Equation 5.4, becomes the following linear discrete time transfer function:

$$Y_{av}(z) = Y_{MC}(z) \cdot (C - MC) \tag{5.5}$$

Finally, utilising the Z transform properties, a general availability model can also be given in form of a time invariant, linear difference equation in Equation (5.6):

$$y_{av}(k) = -\sum_{m} d_m \cdot y_{av}(k-m) + \sum_{n} e_n(\cdot C(k-n) - MC(k-n)) \tag{5.6}$$

where the coefficients a_i, b_j are to be found by the identification algorithm detailed below in Section 5.6. Similarly to the performance model, availability dynamic depends on past availability and past clients values that exceeded the max clients MC limit.

5.5 MapReduce Performance and Dependability model

In this section, contrary to the previous sections where we had Single Input - Single Output (SISO) models, in this section we are searching for a Multi Input - Multi Output (MIMO)

model. This means that we desire to quantify at the same time the effects of all the inputs on all the outputs. Following an in-depth analysis of Figures 5.2, 5.3 and 5.4, we can combine Equation (5.1) and Equation (5.4) to reach the general performance and availability MIMO structure of the MapReduce system, given in Figure 5.1:

$$\begin{aligned} Y_{rt}(z) &= Y_C(z) \cdot sat_0^{MC}(C) + Y_N(z) \cdot N \\ Y_{av}(z) &= Y_{MC}(z) \cdot sat_0^{\infty}(C - MC) \end{aligned} \tag{5.7}$$

One thing to notice in Equation (5.7) is that when we combine the two models we need to add an input saturation to the cluster size C input for the performance model, since performance is only influenced by the number of clients that are below or equal to the max clients level. While, contrary to the previous, availability is influenced by only the number of clients that are above this level. Using the same reasoning as in the case of the availability model, if we take the control design decision to always keep $C \geqslant MC > 0$, we can remove both of the input saturations from the model and we can transform Equation (5.7) into a linear one in Equation (5.8), that contains only discrete time linear models.

$$\begin{aligned} Y_{rt}(z) &= Y_C(z) \cdot MC + Y_N(z) \cdot N \\ Y_{av}(z) &= Y_{MC}(z) \cdot (C - MC) \end{aligned} \tag{5.8}$$

Without making any assumptions on the orders of Y_C, Y_N, Y_{MC} discrete time transfer functions, we can use inverse Z transform properties to convert Equation (5.8) into a more general linear difference equation, as presented in Equation (5.9).

$$\begin{aligned} y_{rt}(k) &= -\sum_m d_m \cdot y_{rt}(k-m) + \sum_n MC(k - \tau_{rt_{mc}} - n) + \sum_o f_o \cdot g_p \cdot N(k - \tau_{rt_n} - o) \\ y_{av}(k) &= -\sum_i a_i \cdot y_{av}(k-i) + \sum_j b_j(\cdot C(k-j) - MC(k-j)) \end{aligned} \tag{5.9}$$

As we can see in Equation (5.9) when it comes to MIMO systems, having multiple transfer functions that describe systems dynamic becomes difficult to deal with at control design time. Therefore, control theoreticians prefer the general state space formulation to treat MIMO systems.

The general equation of a general discrete linear system in state space form is given in Equation (5.10):

$$\begin{aligned} x_{k+1} &= A_d \cdot x_k + B_d \cdot u_k \\ y_k &= C_d \cdot x_k \end{aligned} \tag{5.10}$$

where $y_k = \begin{pmatrix} y_{rt} \\ y_{av} \end{pmatrix}$ is a vector containing the system outputs, $u_k = \begin{pmatrix} N \\ C \\ MC \end{pmatrix}$ is vector containing the system inputs and $x_k = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{pmatrix}$ is a vector that contains n number of system states.

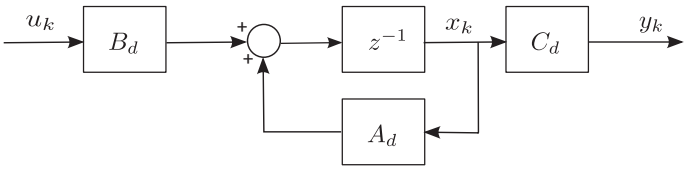


Figure 5.7: General MIMO state space model structure

The structure of the general model in state space form is presented in Figure 5.7.
The advantages of such a model formulation are its generality and scalability. Regardless of the number of inputs-outputs a system model may have, any linear system can be described in form given in Equation (5.10). Therefore, it provides us a general approach to treat any system in a unified manner, as using this notation SISO and MIMO can be formally treated equally.

The dynamics of the system in this case is contained in the matrices A_d, B_d, C_d:

- A_d is an $n \times n$ matrix that captures the dynamics of how the system states evolve over time. n is the number of states in the model.
- B_d is an $n \times 3$ matrix and provides the influence of the inputs C, N, MC on the evolution of the system states.
- C_d is an $2 \times n$ matrix that gives the relationship between the system states and the system outputs.

Let us take Equation (5.8) for example and rewrite it into the following matrix:

$$\underbrace{\begin{pmatrix} y_{rt} \\ y_{av} \end{pmatrix}}_{y(k)} = \underbrace{\begin{pmatrix} Z_C(z) & Z_N(z) & 0 \\ 0 & -Z_{MC}(z) & Z_{MC}(z) \end{pmatrix}}_{H(z)} \cdot \underbrace{\begin{pmatrix} N \\ C \\ MC \end{pmatrix}}_{u(k)} \tag{5.11}$$

If we note by $H(z)$ the matricial form of the transfer functions, the connection between Equation (5.8) and Equation (5.10), namely between the transfer functions Z_C, Z_N, Z_{MC} and the matrices A_d, B_d, C_d, can be made using the following control theoretical formula:

$$H(z) = C_d(zI - A_d)^{-1}B_d \tag{5.12}$$

where I is the identity matrix of the same size as A_d. Finding these system matrices for a particular system can be done in multiple ways. There are many identification techniques that calculate them directly from the identification data. Moreover, in case of linear systems there are multiple automated tools (see Matlab's 'tf2ss' function) that can convert between the transfer function formulation in Equation (5.8) and the state space formulation of Equation (5.10).

In our case, after performing the conversion to state space, the matrices A_d, B_d, C_d have the following structure:

$$A_d = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & a_{33} & 0 \\ 0 & 0 & 0 & a_{44} \end{pmatrix}, B_d = \begin{pmatrix} b_{11} & 0 & 0 \\ b_{21} & 0 & 0 \\ 0 & b_{32} & 0 \\ 0 & b_{42} & b_{43} \end{pmatrix}, C_d = \begin{pmatrix} c_{11} & 0 & c_{13} & 0 \\ 0 & 0 & 0 & c_{24} \end{pmatrix}$$

As we can see in Equation (5.10), the particularity of state space models compared with transfer functions is that, system outputs are not described only in function of the input variables u_k but also in terms of other, intermediary variables called state variables x_k. While for physical system these variables can be related most of the time to basic physical processes properties such as energy, mass. For software systems it is difficult to make this connections, because of lack of physics behind them. Nonetheless, arriving to such form is indispensable to be able to use the vast array of advance control theoretical algorithms developed for MIMO systems.

5.6 Model identification

5.6.1 Off-line identification of the model parameters

Identifying the parameters of a mathematical model from observed data is a very well known technique in control theory. One can check for example [42] for a detailed review of different methods and identification algorithms. From our point of view it can be briefly summarized by Figure 5.8.

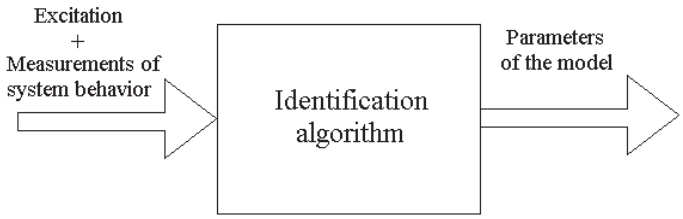


Figure 5.8: General identification procedure

The parameters of the mathematical model are found by giving the excitation signal and the system's output measurements as inputs to identification function, and the function automatically returns the parameters. We can see that these are simple steps to follow, therefore the whole parameter identification procedure can be easily automated.

In cases when we have a MIMO model, all that needs to be done is to identify a separate model from each input to each output. Namely the inputs are varied one by one, while keeping the others fixed, and the outputs of the system are measured. After getting the *(input,output)* training data an existing linear regression function (e.g the Matlab function *'procest'*) is used

to obtain the model parameters. The desired order for the model can be specified in advance or chosen automatically and the function returns the model parameters accordingly. The validity of the identified parameters depends upon the linearity of the system. In practice, if we detect that we leave the linear region and the model accuracy decreases under a given limit, then the automatic identification procedure is repeated or on-line parameter adaptation is performed.

In case of linear systems, an identified model of high order it is not compulsory for a good estimation from a control point of view [9]. If the order is too high the corresponding coefficients can have a negligibly small value (eq. 10^{-5}). Meanwhile, the higher the chosen model order the more complicated the control design can become. Therefore, and without any lose of generality, the infinite summations in equations (5.6) and (5.3) can be replaced by first or second order functions. The order of the functions can be easily inferred from systems response to the excitation input signal. Furthermore, the equations given in the following sections can be easily applied to other systems, with only the re-identification of the equation parameters. Moreover, if the user wants, it can impose higher order polynomials for approximation and the algorithm will work the same way in finding all the coefficients.

There are many different identification algorithms developed in the speciality literature [41]. We chose the prediction error method as our identification algorithm [70]. This method has been shown to provide optimal results (minimal covariance matrix) in the case when the chosen model order fits well the true system [41].

In order to run the identification algorithm, all we need is to collect is the training experimental data. This training data consist of an input excitation for the MapReduce system and the measurements of system's response to this excitation. The most simple excitation has the shape of a step: namely during the course of the experiment the input signal has a single sudden change (increase or decrease) and then it remains constant for the remaining time.

In the following, the formal formulation of the identification problem is presented:

- The system's response $\{y_m(1) \cdots y_m(n)\}$ to the input set $\{u(1) \cdots u(n)\}$ is collected.
- Based on the shape of the output data the desired model structure is selected, for example:
$$y(k+1) = -\sum_{i=1}^{n} a_i \cdot y(k+1-i) + \sum_{i=0}^{m} b_i u(k-\tau-i) \tag{5.13}$$
- We run the identification algorithm to find the parameters, a_i and b_i, with which our model best approximates the system behaviour.

Let us now formally define the identification algorithm. Using the vectorial notations $\theta = [a_1, a_2, \cdots, a_n, b_1, b_2, \cdots, b_m]$ and $\phi(k) = [-y(k), \cdots - y(k+1-n), u(k-\tau), \cdots, u(k-\tau-m)]$ Equation (5.13) can be easily reformulated to Equation (5.14)

$$y(k+1) = \theta^T \cdot \ \phi \tag{5.14}$$

where the unknown system parameters are encapsulated into the θ vector.

Therefore the objective at hand is how to identify the θ parameter vector. In case of the PEM algorithm the question is posed as a cost function minimisation and is solved using numeric optimisation techniques, such as the Quasi-Newton method. A simplified form of the optimization criteria is given in Equation (5.15)

$$J = \min_{\theta} \sum_{k=1}^{N} \epsilon^2(k) \tag{5.15}$$

where $\epsilon = y_m(k) - y(k) = y_m(k) - \theta^T \cdot \phi(k-1)$ is the prediction error, that is to say the difference between the predicted $y_m(k)$ and measured outputs $y(k)$, and N is size of the identification data. In case of linear systems, an analytical solution for the minimisation criteria defined above can be easily calculated, giving us the searched θ parameter vector.

5.6.2 On-line adaptation of the model parameters

One of the limitations of the model previously defined is that, it is the most accurate only around a certain operating point. As a result of the linearisation the model accuracy gradually decreases as we get further away from this point, the magnitude of the accuracy decrease being proportional to the non-linearity of the system. However, we can improve the model accuracy through on-line model adaptation techniques. The idea behind these techniques is to keep the previously found model structure, but identify its parameters (θ) on-line.

One of the most used such techniques in practice, due to its fast convergence property, is the Recursive Least Square Estimator [10]. The intuition behind the algorithm of finding the model parameters on-line can be summarised like this:

- First the model structure is determined based on off-line identification techniques, described in the previous section.
- Keeping the previously determined structure fixed, with each new measurement we update the model parameters, using a recursive least square estimator for example. The objective of the adaptation algorithm is to periodically minimise the error between our modelled dynamics and the real system response. Although the model accuracy might be poor initially, it improves over time with each new measurement and converges quickly to the real system.

In our case, as our system model is linear, a least square recursive estimator is sufficient for parameter adaptation. The steps of updating the parameters of a linear model using this technique are the following:

Step.1 Initialize the model parameters θ, the estimator gain K and the covariance matrix P. K tells how much the new measurements affect the model parameters. While P is the

inverse of the measurement noise covariance, assigning more weight to measurements with low variance.

Step.2 Update the model parameters, based on the difference of the new measurement $y_m(k)$ from the previous prediction, using the following formula:

$$\theta(k) = \theta(k-1) + K(k)\left(y_m(k) - \phi^T(k)\theta(k-1)\right) \tag{5.16}$$

Step.3 Recalculate the estimator gain K that gives the relative importance of the current measurement, with respect to prior parameter estimates:

$$K(k) = P(k-1)\phi(k)\left(\lambda + \phi^T(k)P(k-1)\phi(k)\right)^{-1} \tag{5.17}$$

K is a proportional gain that indicates how much correction should be taken based on the new error measurement, taking into consideration the measurement noise encapsulated in P. λ is an exponential forgetting factor, assigning exponentially less weight to older measurements.

Step.4 Update the covariance matrix P. If P is large, it means that the parameters are estimated to change significantly, as a result the estimator gain K needs to be large. If P is small, it means that the model parameters don't vary much any more, therefore the gain K becomes smaller to minimise the changes caused by new measurements.

$$P(k) = \lambda^{-1}\left(I - K(k)\phi^T(k)\right)P(k-1) \tag{5.18}$$

Step.5 Return to Step.2.

It has been mathematically proven that as k increases over time, meaning more and more measurements are available, the difference between the measurements and the model prediction is reduced and the algorithm converges to the real values of system parameters. One of the big advantage of this method is that it identifies the model parameters at runtime, without the need for previous experimental data.

For a more detailed description of how to use on-line adaptation techniques for a MapReduce system in practice, please see the Master Thesis of Sophie Cerf [17]. For more information regarding the state of the art on-line adaptation techniques, such as the one presented above, please see [10].

5.7 Summary

In this chapter, first a general methodology of assigning a measurable dynamics to software systems is presented. Then the input-output variables of the general MapReduce dynamic model are defined. The modelling of the relationships between each input and output are analysed in detail. Combining the results of these examinations, the structure of the general MIMO MapReduce model is proposed. Finally the techniques of finding the model parameters

are discussed in detail. In the following Chapter 6 these models are used to design and analyse several feedback control laws for ensuring MapReduce performance and availability. The experimental evaluation of the presented models is given in Section 7.2.

Chapter 6

Control of MapReduce Performance and Availability

Contents

6.1 Classical control 56
6.1.1 Control architecture 56
6.1.2 Feedback control 57
6.1.3 Feedforward control 59
6.2 Event-based control 61
6.2.1 Control architecture 61
6.2.2 Event-based feedback control 63
6.2.3 Event-based feed-forward control 65
6.3 Constrained optimal control: *MR-Ctrl* 66
6.3.1 Control architecture 66
6.3.2 *MR-Ctrl*- Optimal control with constraints 67
6.3.3 Improving control robustness through integral action 70
6.3.4 Compensating for actuation delays 70
6.3.5 *MR-Ctrl*- observer 71
6.4 Summary 72

In this chapter the on-line control algorithms for ensuring the performance and dependability of a MapReduce cluster are developed. First, in Section 6.1 we introduce a control architecture that ensures MapReduce performance through cluster scaling, based on a classical time based PI and feedforward controllers. In the second part we further adapt the previous time based control architecture to the cloud environment through event-based techniques. These novel event-based proportional-integral and feedforward controllers are presented in Section 6.2. Finally, in Section 6.3 the unified optimal control framework, named *MR-Ctrl*, is developed. It can ensure at the same time both the performance and availability of a MapReduce service, meanwhile explicitly minimising costs.

6.1 Classical control

6.1.1 Control architecture

In this first control instance, our objective is to control the performance of the MapReduce system. We have as control input the cluster size $N(k)$ and as exogenous, disturbance input $C(k)$ the workload size. Our output is the average response time $y_{rt}(k)$.

The complete schema of our control architecture for MapReduce performance control, is presented in Figure 6.1. The variables used are defined in Table 6.1 below the figure.

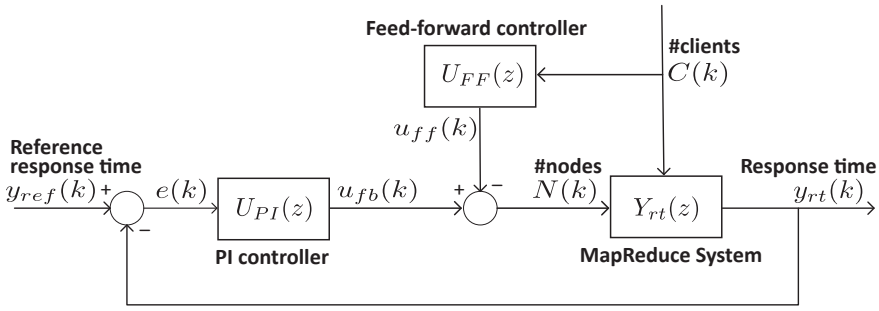


Figure 6.1: Classical control architecture

y_{ref}	Reference average response time set in the SLA.
y_{rt}	System output - average response time of client interactions.
N	Control input - cluster size.
u_{fb}	PI component of the control input.
u_{ff}	Feedforward component of the control input.
e	Difference between the reference and measured response times.
C	Disturbance input - workload size.
U_{PI}	Discrete time transfer function of the PI feedback controller.
U_{FF}	Discrete time transfer function of the feedforward controller.

Table 6.1: Definition of control variables.

As we can see in Figure 6.1, our cluster size control input is made up of the sum of two controllers, a feedback and a feedforward one. The role of the PI feedback controller is to ensure good disturbance rejection and the robustness of the control loop to any modeling and environmental uncertainties. Moreover, as we can accurately measure the workload size on-line, we design a feedforward controller that can use this information in order to assure a faster controller response. It can do this by counteracting the disturbance before its effects can be measured on the output. This type of control architectures have well proven their efficiency in the different domains, from the early 1900's, and in the following we demonstrate

that they can be successfully applied to this type of systems as well. The next sections discuss these different control strategies in detail.

6.1.2 Feedback control

Feedback control is everywhere from the automotive, robotic, energetic industries to the microelectronics one. Still, with a few exceptions, it is far from a mainstream tool utilised in the design of software systems. Nevertheless, reactive control algorithms, which can be considered a different approach to feedback control, have been extensively used for adaptation in software engineering. These dynamic adaptation techniques are generally based on the MAPE-k loop. For a detailed comparison between feedback control and MAPE-k the reader can further check Section 2.6. Based on this comparison we infer that feedback control architecture does provide several advantages to the traditional MAPE-k approach. These advantages are briefly detailed below:

- The feedback control decisions are based on a dynamic model of the system in comparison to the static model of the MAPE-k. The multiple decades of experience in controlling physical systems showed that an in-depth knowledge of the system behaviour over time is essential to decide on when and how to control.
- While formal proofs are difficult in case of the MAPE-k loop, feedback control can provide easily provable assurances in term of control stability, efficiency and robustness.
- The formal, mathematically rigorous approach of feedback control improves upon the generality of the control solutions which, as a result, can be easily duplicated from one system to an other.
- There are multiple feedback control architectures that have already proved their usefulness and robustness in case of many physical systems. This means that using this formalism we benefit of a multitude of control approaches that have already been extensively tested in a large variety of industrial scenarios.

One of the most used control techniques in industrial practice is the PID (Proportional-Integral-Derivative) controller. In fact, this type of control loops are used in almost 90% of industrial processes. The reason why it is one of the most utilised controllers is that, even with a relative simple structure, it can still satisfy most control objectives in practice, with a high degree of reliability. The intuition behind this control structure is the following:

- The proportional term means that the controller reacts proportionally to the error, namely the size of the current difference between the expected output and the measured one. It is a well know property of proportional control that it cannot ensure complete convergence to the reference output value, even after the system stabilises to a steady state. This is because if this error becomes very small, the controller will not react to it anymore. As a result, in most of the cases in practice it is utilised together with the integral term that can compensate for this.

- The integral component calculates the control based on error evolution over time, by calculating the integral of all past errors. It adds a very robust steady state tracking property to the P control. It overcomes the previously described shortcoming and ensures 0 steady state control error.
- The derivative term looks at the slope of the error change, therefore it calculates the control based on a prediction of the future error. It adds an extra tuning option for the control and can improve on control convergence times. However, since this term is very sensitive to high frequency noise in error measurements, it is frequently omitted in practice if we have a noisy error measurement.

As we have noisy error measurement, for our system a PI controller is chosen. It is well proven that for such a system (i.e. a first order system with deadtime - see Equation (5.2)) a PI controller is sufficient even if the system is complex, with eventually higher order dynamics [29]. Furthermore, a feedback controller with integral action offers several well known advantages. It compensates for the errors that arise from model imperfections and has well-proven disturbance rejection properties for even unmeasured and un-modelled disturbances.

The discrete time transfer function of a general PI controller is given in Equation (6.1):

$$U_{PI}(z) = K_p + \frac{K_i \cdot z}{z - 1} \tag{6.1}$$

where the proportional gain (K_p) and the integral gain (K_i) are control tuning parameters. Applying the invers Z-transform to Equation (6.1), we can calculate the discrete difference equation form of the controller, shown equation (6.2):

$$u_{fb}(k) = u_{fb}(k-1) + (K_p + K_i) \cdot e(k) - K_p \cdot e(k-1) \tag{6.2}$$

where $k \in \mathbb{N}$ is the sampling instance.

There are numerous theoretical and practical tools developed for finding the controller tuning variables (K_p, K_i). For a detailed description the reader should check [29].

We opted for a technique called loop shaping, which implies the shaping of the form of the open loop transfer function for closed loop stability, performance and robustness. In fact robustness and performance are inversely proportional. The larger the bandwidth of the loop is, the better the performance (quicker tracking and disturbance rejection). However, as we increase the bandwidth we decrease the system phase and gain margins, therefore decrease the control robustness to modelling errors or changes in the system dynamics. The trade-off between the two is usually decided by first ensuring performance requirements, such as response time, overshoot, while maximising robustness. Matlab, for example, provides simple to use visual tools such as "pidtool" [46] which allows for choosing the performance and robustness levels using a couple of simple sliders.

In our case the control tuning parameters (K_p, K_i) are determined by first ensuring closed loop stability and no overshoot in both system output and in control input. As in practice we

would like to avoid a highly aggressive controller, the controllers response to the disturbance is somewhat slow. The reason why the lack of overshoot in control as well is desirable is that, we wish to minimize the number of cluster reconfigurations, therefore adding and removing nodes over a short period of time is not suitable.

6.1.3 Feedforward control

Feedforward control in automatics is about modelling the effects of disturbances on the system and using this knowledge to cancel their impact on the system. This is done by initiating the inverse effect on the system through the control inputs. It can only be used in cases when the disturbances can be measured on-line. The feedforward controllers purpose is to create a control signal that passes through the system at the same time as the disturbance and cancels out its effect, thus keeping the system outputs stable. Feedforward control can considerably improve control responsiveness. Moreover, in case of large delays in the disturbance effect, the benefits of a fast feed forward controller become even more visible, as this delay will propagate to the output measurements. Which implies that a feedback controller that is based on output measurements will react to any changes later as well. Meanwhile, a feedforward controller can react at the instant the disturbance arrives and pro-actively reject the effect of the disturbance changes, before their effect can be even measured on the output.

Nevertheless, the effectiveness of the feed forward controller depends entirely on the accuracy of the identified model. Therefore, to improve robustness it is generally used together with feedback control that can correct for modelling errors. Of course, if the model is very accurate the net effect on the response time should be zero, but because of the inherent model uncertainties this is rarely the case in practice.

Furthermore, although feedforward control can be considered a predictive control it must not be confused with the demand prediction algorithms used in computing system [50]. The feedforward model does not predict how the demand varies over time. Instead, it predicts what the dynamic effect of a change in the demand size is on the respective system.

In our case, let us first recall the performance model given in Figure 5.5 (p. 45). Our system output is influenced by the cluster size (N) and the workload size (C). The workload size (C) can be treated as a measurable disturbance to the system. Since we can measure on-line the number of clients in the system, we can design a feedforward controller to reject the effects of workload size variations upon response time.

Our feedforward controller is determined using the standard feedforward formula given in Equation (6.3):

$$U_{ff}(z) = -Y_N(z)^{-1} \cdot Y_C(z) \tag{6.3}$$

where $U_{ff}(z)$ is the discrete time controller and $Y_N(z)$, $Y_C(z)$ are the discrete time transfer functions given in Equation (5.1) (p. 44).

If we consider $Y_N(z)$ and $Y_C(z)$ to have a first order plus deadtime (FOPTD) structure,

then Equation (6.3) can be rewritten as Equation (6.4):

$$U_{ff}(z) = z^{(\tau_{rt_n} - \tau_{rt_c})} \frac{b_c \cdot (z + a_n)}{b_n \cdot (z + a_c)} \tag{6.4}$$

One thing we can notice from Equation (6.4) is that, a feedforward controller is not always realisable in practice. For example, if the workload size dynamics $Y_C(z)$ is faster than the cluster size dynamics $Y_N(z)$, the feedforward action will always be late. Therefore, it cannot counter the disturbance effects while it is happening. To better understand this, let us consider that the delay in cluster size variation τ_{rt_n} is larger than that of the disturbance τ_{rt_c}. In this case the equation becomes acausal, meaning that if we transform the equation into the difference equation form, then our model will have future values in the output dynamics calculation. One solution in such cases is to use a lead-lag feedforward control structure, where we have a large lead component to give an initial jolt to control input, to catch up to the disturbance effect [62].

Furthermore, if we consider the traditional control profile of a feedforward controller in response to a workload increase, its general behaviour is to initially add a larger number of nodes and than to logarithmically decrease to the new stabilizing level. Although this approach compensate exactly for the changes in the system, such a behaviour is not always desirable in case of cloud systems as there is cost penalty to pay if you remove nodes shortly after their addition. Nevertheless, if performance is prioritized over cost, as is the case of a strict performance control SLO, then this behaviour is acceptable.

Nevertheless, if we have relaxed performance constraints, with minimal resource usage being a top priority, then this behaviour is not desirable. Therefore we have modified the traditional feedforward response to act as a feedforward gain and add directly the nodes required after the absorption of the initial shock. Although the performance of the feedforward controller is decreased, we have considerable gains in control cost. The feedfoward control is calculated in this case based only on the gains of the disturbance and system models. Consequently, if we remove the control dynamics from Equation (6.4) we are left with a static control gain calculated from the division of the gain of the disturbance model with the process one.

$$U_{FF} = -\frac{b_C}{b_N} \tag{6.5}$$

The intuition for this last control technique can be approached, from the software engineering point of view, by answering the following question a system engineer might pose:
If we have x clients that arrive/leave how many nodes should I add/remove to compensate for their steady state effect on system performance?

The answer to this question being $-\frac{b_C}{b_N} \cdot x$.

6.2 Event-based control

6.2.1 Control architecture

The event-based control architecture presented in this section was developed to overcome the following shortcomings of the classical control presented in the previous section:

- Control profile:

  When building classical control algorithms we are traditionally interested mostly only at the form of the output signal and put only magnitude limitations on the control one. The control profile is mostly of interest in cases when there is a desire to improve the lifetime of physical actuators. Somewhat analogously to the latter, in the case of cloud elasticity, the form of the control signal can be just as or more important then the output signal. The reason for this being, that the service cost is a function of the control profile. Namely the addition and removal of resources has energetic and financial costs. Furthermore, one would like to minimise the number of changes in the control signal or in other words the number of cluster reconfigurations. This allows the system to be more predictable and also for the different optimization algorithms (schedulers) that are running besides our control to converge.

- Control reactivity:

  A somewhat unusual requirement from the point of view of traditional control systems is that, in cloud scenarios there are cases when one would like to reduce controller reactivity, by introducing a deadband. In fact, there is need for the possibility to define certain limits between which the controller would not react at all. For example, there are practical scenarios when one would like the controller to react only if the output differs from the reference with a significant amount. Or in the case of a feedforward controller one would like if the feedforward component reacted only if the workload change is sufficiently large.

Therefore, the question arises: how can we introduce such thresholds together with the dynamic control techniques described in the previous section and address the previously listed shortcomings?

Before providing an answer to this question, let us introduce a novel controller type called event-based controllers. These controllers have emerged recently as a viable alternative to periodic, time based controllers when it comes to handling constraints on the number of actuations, limited communication or computation bandwidth, constraints on power consumption, etc [45, 67].

The basic idea behind event-based control theory is that, it is not mandatory to calculate and update the control signal with every new measurement but instead, the control is recalculated only when the error, between the system output and reference value, crosses a certain threshold since the last actuation. This concept is illustrated in Figure 6.2, where a

simple example is shown to clarify the difference between the control instants of time-based and event-based controllers. For a more detailed view of event-based control theory see [8, 45, 67].

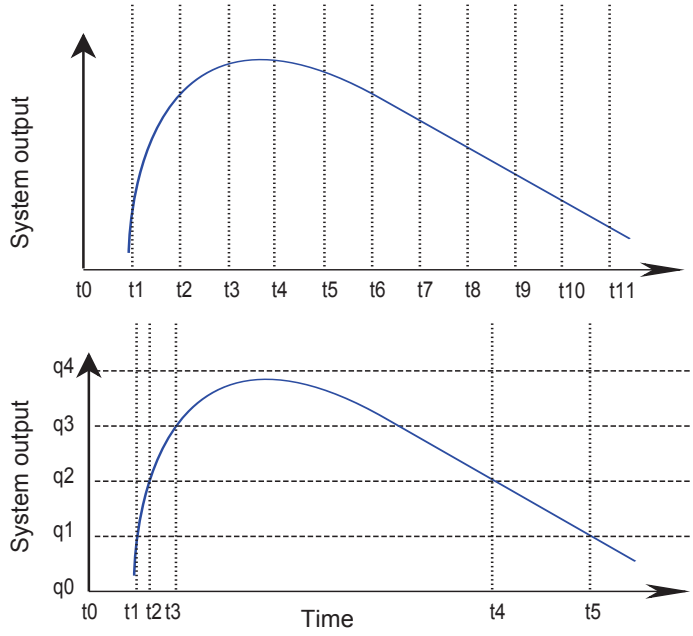


Figure 6.2: Difference between time-based and event-based control instants

Figure 6.2 shows that while the control instants of time-based control follow a fixed time period $t_{i+1} - t_i$, for the event-based case the control is calculated only when the system output changes more than a fixed threshold value ϵ. The Y axis in this simple case can be any feedback signal used for control purposes. A simple analogy to the event-based train of thought is the case of a server process that needs to collect the data from its child process. Instead of polling the child processes at fixed periodic instant, an event is generated in the parent process when one or more of its parameters of the child process are outside predefined limits.

For an other example, let us look at the case of power consumption reduction in embedded control systems. This decrease in energy usage is achieved by putting the system into a low energy state when the changes from the set-point are within certain limits. While, in the case of cloud systems, this energy consumption caused by the control calculations is negligible, we will show that the event-based formulation can be very useful to improve upon the drawbacks concerning the control profile and reactivity of the time based control version.

Now, to answer the question posed at the beginning of this section, we have developed an event-based control framework adapted to the MapReduce cloud scenario. Through it, we improve upon the control profile and reactivity of the control architecture presented in the previous section, by switching from time-based to an event-based control strategy.

The complete schema of the event-based control architecture is presented in Figure 6.3. The input/output variables used in the figure are the same as in the previous section and are

defined in Table 6.1 (p. 56). Similarly to Figure 6.1, we have two inputs: the control input $N(k)$, which is the number of nodes in the cluster and the exogenous input $C(k)$, which is the workload size, and one output, that is the response time $y_{rt}(k)$.

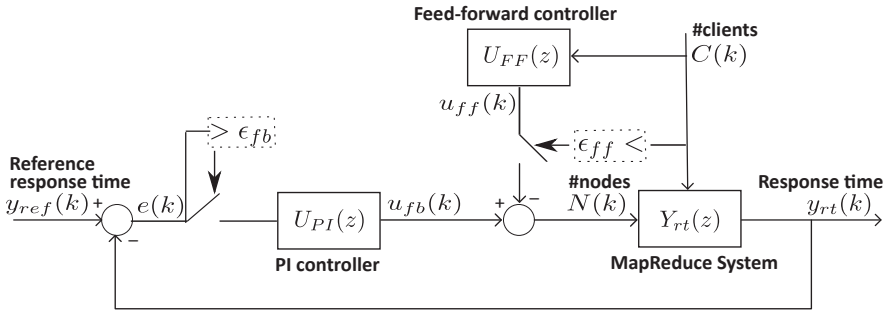


Figure 6.3: Event-based control architecture

These improvements are done with the use of the extra tuning parameters introduced by the event-based controller. namely the event functions: ϵ_{fb}, ϵ_{ff}. These two function can be used to design a dynamic version of the static thresholding based (if-then) approaches, currently utilised for automatic cloud elasticity solutions. Furthermore, using these functions one can define a dynamic band around the operating point, where control is not calculated at all. The dynamic nature of our control approach comes from the fact that, the detection levels of the event functions are defined not in terms of a certain value, magnitude of the output signal. Instead, they are defined on the change magnitude of the signal, which in most cases is an error signal.

Traditionally the biggest challenges in event-based control techniques is the definition of the detection levels used in the event functions. An analogous problem for computer scientists is finding the static thresholds that ensure the desired system performance. Nevertheless, we will show that in the case of computing system, we can make use of several common statistical metrics to define these detection levels.

6.2.2 Event-based feedback control

Whereas a classical PI controller was presented in Section 6.1.2, an event-based version is developed here. The considered closed-loop system is represented in Figure 6.3. The proposed approach is based on an original event-based PI controller, which setup was suggested for the first time in [7] and then improved in [22].

By an ***event-based feedback controller*** we think of a set of two functions:

- an *event function* ϵ_{fb}, that indicates if one needs (when $\epsilon_{fb} \leq 0$) or not (when $\epsilon_{fb} > 0$) to recompute the control law.

- a *feedback control law* υ_{fb} and, more particularly in the present case, a PI feedback strategy $\upsilon_{fb}(t) = U_{PI}(z) \cdot e(t)$ where $e(t)$ is the error defined as $e(t) = y_{ref} - y(t)$.

The event function is time-triggered with the sampling period $\bar{h}$ (that is the same as for the corresponding conventional time-triggered PI). On the other hand, the control signal is constant between two successive events

$$u_{fb}(t) = \upsilon_{fb}(t_i) \quad \forall t \in [t_i, t_{i+1}] \tag{6.6}$$

where t_i is a sampling instant (called a *feedback's event*) and, therefore, the length of the sampling intervals $h_{fb} := t_i - t_{i-1}$ is not equidistant in time anymore.

In our case an event is enforced when the *relative error* (the absolute value of the difference between the error of the last sampling and that of the current time instant) crosses a given detection level $\bar{e}_{fb}$, this defines the event function as

$$\epsilon_{fb}(t) := \bar{e}_{fb} - \left|e(t) - e(t_i)\right| \tag{6.7}$$

Several event-based PI strategies were suggested in [22]. The *algorithm with exponential forgetting factor of the sampling interval* is applied here to reduce the impact of the sampling interval increase (which can become huge since there is no boundary in the event-based scheme). The approach is somehow similar to the anti-windup mechanism used in control theory, where the error induced by the saturation has to be compensated. Furthermore, the Tustin bilinear approximation is preferred here (whereas the backward difference approximation was initially used in [22]). The resulting expression (in the z-domain) is

$$\begin{aligned} Z_{PI}(z, h_{fb}) &= K_p + K_i \cdot \delta(h_{fb}) \frac{1 + z^{-1}}{1 - z^{-1}} \\ \delta(h_{fb}) &= h_{fb} \cdot e^{\alpha(\bar{h} - h_{fb})} \end{aligned} \tag{6.8}$$

where h_{fb} is the varying sampling interval (the time between two successive events) and $\delta(h_{fb})$ is the exponentially decreasing sampling interval, α is a degree of freedom to increase/decrease this exponential sampling interval and K_p and K_i are the feedback control parameters. One can refer to [22] for further details.

To define the detection level $\bar{e}_{fb}$, we propose a technique based on the consideration of the natural variance of the system. This builds upon the fact that in the case of computing systems, because of the nature of the aggregating output functions (mean) and because of many possible points of contention that might arise, our output is never stable, but instead will have small oscillations around the steady state. If we use a time-based controller, the controller might react to these small oscillations, that would lead to frequent cluster configurations, something which is not desirable in practice as it would rack up the control costs.

In practice the calculation of this detection level can be done in several ways, the simplest one being to take the maximum value of the output signal in steady state and subtract the mean. Nevertheless, this method should be avoided because it can be skewed by the presence of outliers in the data. We propose the following methodology of determining the detection level $\bar{e}_{fb}$, based on out the output measurement data set $(y_i, i = 1 : n)$:

Step.1 First the data needs to filtered to eliminate any outliers. This can be done easily by calculating the median of the sample and eliminating all values above 2 standard deviations from this median.

Step.2 The sampled mean (Equation (6.9)) is calculated:

$$\mu_n = \frac{1}{n-1}\sum_{i=1}^{n} y_i \tag{6.9}$$

Step.3 Using this mean the sampled standard deviation is computed (Equation (6.10)):

$$\sigma_e = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(y_i - \mu_n)^2} \tag{6.10}$$

Step.4 This standard deviation value can be considered a way of measuring how far the average data values lies from the mean. Therefore, we can consider this value or a multiple of it as our detection limit $\bar{e}_{fb} = \sigma_e$. By this we can ensure that the feedback controller doesn't react to the natural variation of the system, which considerably reduces the number of cluster reconfigurations. This standard deviation calculation can be done off-line or on-line in recursive manner.

6.2.3 Event-based feed-forward control

In order to pro-actively reject the effect of a change in the workload size C, before it can impact our system output y_{rt}, a fast, feedforward controller $U_{FF}(z)$ is also designed (see Figure 6.3). By analogy with the event-based feedback scheme, an event-based feedforward control is also proposed here. The closed-loop system is depicted in Figure 6.3.
By ***event-based feedforward control*** we mean a set of two functions:

- an *event function* ϵ_{ff} (see the definition above);
- a *feedforward control law* υ_{ff} that is, in the present case, in the form $\upsilon_{ff}(t) = U_{FF}(z) \cdot C(t)$, where $C(t)$ is the disturbance (the workload size).

As before, the event function is time-triggered with the sampling period $\bar{h}$ while the control signal is constant between two successive events

$$u_{ff}(t) = \upsilon_{ff}(t_j) \quad \forall t \in [t_j, t_{j+1}] \tag{6.11}$$

where t_j denotes a *feedforward's event* and $h_{ff} := t_j - t_{j-1}$ is the length of the (varying) sampling intervals.

The control strategy is determined using the standard feedforward formula

$$U_{FF}(z, \bar{h}) = -U_N(z, \bar{h})^{-1} \cdot U_C(z, \bar{h}) \tag{6.12}$$

However, if we consider the traditional version of a feedforward controller, its general behaviour is to initially add a larger number of nodes to counteract the shock on the system and then to slowly decrease to the new steady state level. Although this approach compensate exactly for the dynamic changes in the system, such a behaviour is not desirable in the case of cloud systems as there is cost penalty to pay if you remove nodes shortly after their addition. Therefore we've chosen a modified feedforward response that acts as a feedforward gain and adds directly the nodes required after the absorption of the initial shock. Although the performance of the feedforward controller is decreased, we have significant gains in control cost. The feedfoward gain, as given in Equation (6.13), is calculated based on the division of the disturbance model gain with that of the process one.

$$U_{FF}(z) = -\frac{b_C}{b_N} \tag{6.13}$$

Note also that, whereas the feedback controller (U_{PI}) is not required in the ideal case, here it is compulsory to keep it to ensure robustness of the closed-loop system. In the feedforward controllers case an event is enforced when the *relative disturbance* crosses a given detection level $\bar{e}_{ff}$, namely the change in the workload is significant enough. This event function is defined as:

$$\epsilon_{ff}(t) := \bar{e}_{ff} - \left|C(t) - C(t_j)\right| \tag{6.14}$$

where t_j is the last sampling instant. For finding the detection level $\bar{e}_{ff}$ we can use an analogous approach as for the definition of the feedback detection limit $\bar{e}_{fb}$ and look at the standard variation of the workload size to calculate the detection level $\bar{e}_{ff} = \sigma_C$.

6.3 Constrained optimal control: *MR-Ctrl*

6.3.1 Control architecture

In this chapter we design *MR-Ctrl*, an optimal controller, able to deal with multiple, contradictory objectives. As our MapReduce model has two outputs, *MR-Ctrl* will assure at the same time both the response time and the availability limits specified in the SLA, while explicitly minimizing resource utilization and hence costs.

The complete schema of the control architecture is presented in Figure 6.4. All the variables used in the figure are defined in Table 6.2. For more details regarding the implementation of the control framework one can check [13]. As defined in the general model structure given in Section 5.2.2 (p. 38), we consider the MapReduce system having two inputs (concatenated in the two dimensional vector u), one exogenous uncontrollable disturbance input C and two outputs (concatenated in the two dimensional vector y). Vector u is made up of control inputs cluster size N and max clients number MC, while the vector y contains the response time y_{rt} and availability y_{av} output metrics.

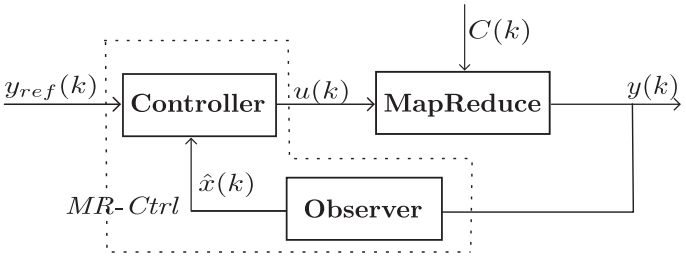


Figure 6.4: *MR-Ctrl* : Optimal control architecture

$y_{ref} = \begin{pmatrix} y_{rt_{ref}} \\ y_{av_{ref}} \end{pmatrix}$	Reference - response time and availability set in the SLA.
$y = \begin{pmatrix} y_{rt} \\ y_{av} \end{pmatrix}$	Measured system output - response time and availability.
$u = \begin{pmatrix} N \\ MC \end{pmatrix}$	System control input - cluster size and the maximum number of clients.
C	Disturbance - Number of clients trying to connect to the system.
$\hat{x}$	State estimate- reconstructed internal behavior of MapReduce.

Table 6.2: Definition of control variables.

6.3.2 *MR-Ctrl*- Optimal control with constraints

In this section we detail the general theoretical framework that is the basis of *MR-Ctrl*.
Let us first consider the general discrete linear system (with sampling period T_s):

$$\begin{aligned} x_{k+1} &= A \cdot x_k + B \cdot u_k \\ y_k &= C \cdot x_k \end{aligned} \tag{6.15}$$

and some prediction horizon T_p, which is a multiple of the sampling period $T_p = n \cdot T_s$.
At each time instant $t = k \cdot T_s$, *MR-Ctrl* recomputes the control U_k that minimises the following criterion:

$$J = \min_{U_k} \{(Y_k - Y_{ref})^T \cdot Q \cdot (Y_k - Y_{ref}) + U_k^T \cdot R \cdot U_k\} \tag{6.16}$$

subject to contraints $0 < M_C \leq C$ and $N < N_{max}$, where the superscript T stands for the transposed vector or matrix.

$$U_k = \begin{pmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+n} \end{pmatrix}, Y_k = \begin{pmatrix} y_{k+1} \\ y_{k+2} \\ \vdots \\ y_{k+n+1} \end{pmatrix}, Y_{ref} = \begin{pmatrix} y_{ref} \\ y_{ref} \\ \vdots \\ y_{ref} \end{pmatrix}$$

Q and R are weighting matrices that can be used to give more importance to the control action or to the outputs. For example Q can be used to give preference to A_v or R_t in case both objectives cannot be met due to input or cost limitations. R can be used to give a cost penalty on cluster size variations. This allows the controller to prioritize between different control options, to minimise cost.

This optimization problem can be solved by any quadratic programming algorithm and the optimal solution is obtained after a finite number of optimization steps. In our case we use *quadprog*, the quadratic routine in the optimization toolbox of MATLAB [72], which can handle multiple types of control constraints. The *quadprog* routine is very easy to use. First we need to define our optimization criteria, for example in the standard form given in Equation (6.16), and choose our control variables. In order to optimise our criteria function along the control variables u, in case of *quadprog*, the optimization criterion needs to be reformulated to have the following form:

$$J = \min_{u}\{u^T \cdot H \cdot u + 2 \cdot f^T \cdot u\} \tag{6.17}$$

where H and f are constant matrices.

The user has the full freedom in choosing the optimization criterion, the only constraint being that the minimisation criteria to be given in the form shown in Equation (6.17). Therefore, to be able to use the advantages of the well proven optimisation function we need to convert Equation (6.16) into the latter form. To achieve this first, we need to write our system model in a state space form, like given in Equation (5.10) (p. 48). Once we have our state space system model, the transformation of the optimization criterion given by Equation (6.16) can be calculated straight forward in the following manner:

$$\begin{aligned}
y_{k+1} &= C \cdot x_{k+1} = C \cdot A \cdot x_k + C \cdot B \cdot u_k \\
y_{k+2} &= C \cdot x_{k+2} = C \cdot A \cdot x_{k+1} + C \cdot B \cdot u_{k+1} \\
&= C \cdot A^2 \cdot x_k + C \cdot A \cdot B \cdot u_k + C \cdot B \cdot u_{k+1} \\
&\vdots \\
y_{k+n+1} &= C \cdot A^{n+1} \cdot x_k + C \cdot A^n \cdot B \cdot u_k + C \cdot A^{n-1} \cdot B \cdot u_{k+1} + \\
&+ \cdots + C \cdot B \cdot u_{k+n}
\end{aligned} \tag{6.18}$$

First we write the equations, that are just a prediction of our system outputs $Y_k = \begin{pmatrix} y_{k+1} \\ y_{k+2} \\ \vdots \\ y_{k+n+1} \end{pmatrix}$, for n steps ahead (where n is the prediction horizon), based on the current state measurement x_k, current inputs u_k and our system model. The reason we write it in this vectorial form is, because we desire to optimise the control profile for n steps ahead in time. This optimal control profile will ensure that our outputs Y_k will follow a desired trajectory defined with our optimization criteria. However, not all these control inputs $u_{k+1}, u_{k+2}, \cdots, u_{k+n}$ are applied to the system, because we might have unknown disturbances that affect our system during this n future steps. Therefore, in order to be robust to unknown disturbances, at each step k we apply just the first calculated control u_{k+1} , then we update our state estimation x_k

with real system measurements and rerun the optimization algorithm. In this way the control trajectory is continuously optimised at each step and disturbances can be taken into account on-line.

For traditional physical systems , the drawbacks of this type of control are the long control calculation times and the energy usage of the calculations. In case of cloud solutions, not only the energy usage of the control calculation is negligible, but because of the large system time constants even control calculation times, in the order of a few seconds, it become insignificant.

Furthermore, in order to arrive to the form required by the optimization function 6.17, the previous equations in Equation (6.18) needs to be rewritten into a more compact, matricial form given in Equation (6.19):

$$Y_k = \Gamma \cdot X_k + \Delta \cdot U_k \tag{6.19}$$

$$X_k = \begin{pmatrix} x_k \\ x_k \\ \vdots \\ x_k \end{pmatrix}, \Gamma = \begin{pmatrix} CA & 0 & \cdots & 0 \\ 0 & CA^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & CA^{n+1} \end{pmatrix} \text{ and } \Delta = \begin{pmatrix} CB & 0 & \cdots & 0 \\ CAB & CB & \cdots & 0 \\ \cdots & & & \\ CA^nB & CA^{n-1}B & \cdots & CB \end{pmatrix}$$

After replacing Y_k from Equation (6.19) into (6.16) we get:

$$\begin{aligned}
&(\Gamma \cdot x_k + \Delta \cdot U_k - Y_{ref})^T \cdot Q \cdot (\Gamma \cdot x_k + \Delta \cdot U_k - Y_{ref}) + U_k^T \cdot R \cdot U_k = \\
&(x_k^T \cdot \Gamma^T + U_k^T \cdot \Delta^T - Y_{ref}^T) \cdot Q \cdot (\Gamma \cdot x_k + \Delta \cdot U_k - Y_{ref}) + U_k^T \cdot R \cdot U_k = \\
&\cancel{(x_k^T \cdot \Gamma^T \cdot Q \cdot \Gamma \cdot x_k} + x_k^T \cdot \Gamma^T \cdot Q \cdot \Delta \cdot U_k - \cancel{x_k^T \cdot \Gamma^T \cdot Q \cdot Y_{ref}} + \\
&U_k^T \cdot \Delta^T \cdot Q \cdot \Gamma \cdot x_k + U_k^T \cdot \Delta^T \cdot Q \cdot \Delta \cdot U_k - U_k^T \cdot \Delta^T \cdot Q \cdot Y_{ref} - \\
&\cancel{Y_{ref}^T \cdot Q \cdot \Gamma \cdot x_k} - Y_{ref}^T \cdot Q \cdot \Delta \cdot U_k + \cancel{Y_{ref}^T \cdot Q \cdot Y_{ref}} + U_k^T \cdot R \cdot U_k = \\
&U_k^T \cdot \Delta^T \cdot Q \cdot \Delta \cdot U_k + 2 \cdot U_k^T \cdot \Delta^T \cdot Q \cdot \Gamma \cdot x_k - 2 \cdot U_k^T \cdot \Delta^T \cdot Q \cdot Y_{ref} + \\
&2 \cdot x_k^T \cdot \Gamma^T \cdot Q \cdot \Delta \cdot U_k - 2 \cdot Y_{ref}^T \cdot Q \cdot \Delta \cdot U_k + U_k^T \cdot R \cdot U_k = \\
&U_k^T \cdot (\Delta^T \cdot Q \cdot \Delta + R) \cdot U_k + 4 \cdot (x_k^T \cdot \Gamma^T \cdot Q \cdot \Delta - Y_{ref}^T \cdot Q \cdot \Delta) \cdot U_k = \\
&\frac{1}{2} \cdot U_k^T \cdot (\Delta^T \cdot Q \cdot \Delta + R) \cdot U_k + 2 \cdot (x_k^T \cdot \Gamma^T \cdot Q \cdot \Delta - Y_{ref}^T \cdot Q \cdot \Delta) \cdot U_k
\end{aligned} \tag{6.20}$$

The crossed-out parts don't contain the optimization parameter U_k, therefore they can not be optimized here and are irrelevant for the optimization criteria.

Finally, by noting $H = \Delta^T \cdot Q \cdot \Delta$ and $f^T = 2 \cdot (X_k^T \cdot \Gamma^T \cdot Q \cdot \Delta - Y_{ref}^T \cdot Q \cdot \Delta)$, we arrive to the shape needed by *quadprog*, and recalled in Equation (6.17). The of formulation our control problem in the manner presented here has several advantages:

1. There are a vast number of possibilities to define the quadratic optimisation functions, which makes the method widely applicable.
2. The optimal solution is obtained after a finite number of optimization steps.
3. The optimisation procedure works the same, independent of the number of inputs-outputs our target system has and can ensure multiple objectives at the same time.

4. We can implicitly define static and dynamic state and control constraints. Such a static control constraint in our case for example is that the number of nodes N must be smaller then a given maximum number of nodes N_{max}. A dynamic constraint is the fact that our max clients control input MC has no effect on the system if $MC > C$. Therefore, to avoid this behaviour, we define the dynamic input constraint $MC \leq C$.

5. Input delays can be implicitly introduced into the formulation of the criteria function.

6. Using weighting matrices Q, R the trade-off between contradictory objectives can be easily quantified.

7. Cost minimisation objectives, control profile and reactivity, for example minimising the number of actuations, as well as minimising resource usage, can be explicitly introduced into the optimisation criteria.

6.3.3 Improving control robustness through integral action

One of the issues with the standard constraint optimal control approach presented so far is that, since it is based on a model of the system, it is susceptible to modelling errors and constant disturbances that can lead to steady state error.

To make our control approach more robust, we add an integral action into the feedback. To add the integral action to our previously developed control loop, we augment the state vector x_k to get the new state vector $\xi_k = \begin{pmatrix} x_k \\ v_k \end{pmatrix}$, where v_k contains the new integral states. Using this new state Equation (5.10) can be rewritten to Equation (6.21):

$$\begin{aligned} \xi_{k+1} &= \underbrace{\begin{pmatrix} A_d & 0 \\ C_d & I \end{pmatrix}}_{\bar{A}_d} \cdot \xi_k + \underbrace{\begin{pmatrix} B_d \\ 0 \end{pmatrix}}_{\bar{B}_d} \cdot \bar{u}_k \\ y_k &= \underbrace{\begin{pmatrix} C_d & 0 \end{pmatrix}}_{\bar{C}_d} \cdot \xi_k \end{aligned} \tag{6.21}$$

where $\bar{u}_k = \begin{pmatrix} u_k & u_i \end{pmatrix}$ is the original control extended with the error inputs u_i, which are an integral of the error over time. Using the control formulation above we can ensure that our system output converges to reference values set in the SLA even in face of small modelling errors and unmodeled disturbances.

6.3.4 Compensating for actuation delays

One of the crucial issues when designing feedback controllers for cloud software systems are the large actuation delays. When these delays are not taken into consideration our controller may lead to a oscillatory output behaviour, if the actuation delay is larger than the control calculation period.

For example, if at time instant k the controller reacts to a new measurement saying that response time increased and takes a corrective action, scaling up the cluster. At the next time instant $k+1$, we have a new system measurement showing once again that we still have increased response time. If we have a large delay in the actuation then, our previous actuation didn't have time to affect the system output. Now, if delay is not considered in the loop, the controller will react again at each new sampling period instant with a new scale up action. This will be done repeatedly until the actuation delay is passed at time instant $k+d$. But at this time we have already scaled up too much, so now the controller will start scaling down. But, because of the unconsidered actuation delays, we will have the same behaviour as when scaling up, leading to an oscillatory output behaviour.

One of the advantages of the previously presented control mechanism is that, the delay compensation can be explicitly formulated in the formulation of the optimisation criteria. To do these we can reformulate Equation (5.10) into Equation (6.23) to introduce the input delay of τ sampling periods into cluster size input N:

$$\begin{aligned} x_{k+1} &= A_d \cdot x_k + B_d^1 \cdot N_{k-\tau} + B_d^2 \cdot MC_k \\ y_k &= C_d \cdot x_k \end{aligned} \tag{6.22}$$

Notice, that because of the $N_{k-\tau}$ term, in order to be able to arrive at the form required by the optimization function, we need to somehow give the function the previous control inputs $N_{k-\tau}, \ldots, N_{k-1}$ for it to be able to calculate the prediction for n steps ahead. Therefore, through a variable change, we augment the states of the previous model with these values. Let this new state variable be η. Then, Equation (6.23) can be transformed like this:

$$\begin{aligned} \alpha_{k+1} &= A_\alpha \cdot \alpha_k + B_\alpha \cdot u_k \\ y_k &= C_\alpha \cdot \alpha_k \end{aligned} \tag{6.23}$$

$$\alpha_k = \begin{pmatrix} x_k \\ N_{k-\tau} \\ \vdots \\ N_{k-1} \end{pmatrix}, A_\alpha = \begin{pmatrix} A_d & B_d & 0 & 0 & \ldots & 0 \\ 0 & 0 & I & 0 & \ldots & 0 \\ 0 & 0 & 0 & I & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & I \\ 0 & 0 & 0 & 0 & \ldots & 0 \end{pmatrix}, B_\alpha = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ I \end{pmatrix}, C_\alpha = \begin{pmatrix} C & 0 & 0 & 0 & \vdots & 0 \end{pmatrix}$$

6.3.5 *MR-Ctrl*- observer

Having found the matrices H and f obtained above in Equation (6.20), the optimum control law can be computed. At a closer look though, we see that the matrix f contains the internal state of the model x_k, concatenated together in the vector X_k. Since the states x_k are not directly measurable, predictions are obtained from a state estimator. The estimates are computed from the measured output y_k and the applied input u_k with the use of a Luenberger observer [43]:

$$\begin{aligned} \hat{x}_{k+1} &= A_d \cdot \hat{x}_k + B_d \cdot u_k + L \cdot (y_k - \hat{y}_k) \\ \hat{y}_k &= C_d \cdot \hat{x}_k \end{aligned} \tag{6.24}$$

where the variables of the observer are commonly denoted by $\hat{x}(k)$ and $\hat{y}(k)$ to distinguish them from the variables of the physical system from Equation (6.15).

Due to the separation principle, we know that we can define the observer gain L independently from the control, without damaging the overall stability of the system. The poles of the observer $A - L \cdot C$ are usually chosen to converge 10 times faster than the poles of the system. In our case we use the pole placement technique [51] in order to assure this and thus find the L matrix.

6.4 Summary

In this chapter three separate on-line control architectures, for ensuring a MapReduce clusters performance and/or availability, are developed. First, a control architecture based on classical time-based PI and feedforward controllers is developed, to ensure MapReduce performance through cluster scaling. After the careful analysis of this classical control theoretical framework, a novel event-based control framework is introduced, that improves upon the control profile by further minimising the number of cluster reconfigurations and cost, and provides more options in designing the control reactivity. Furthermore a modified feedforward approach is presented that benefits of predictive behaviour of traditional feedforward control, but minimises control cost by reducing the control dynamics. Finally, we develop the optimal control framework *MR-Ctrl*, that ensures at the same time both the performance and availability of a MapReduce service, while explicitly minimising the control cost. Moreover, whenever both objectives can not be met due to cost or resource limitations, *MR-Ctrl* can successfully handle the trade-off between performance and availability according to predefined priorities. The following Chapter 7 provides the experimental evaluation of the modeling and control solutions developed until this point. The experimental evaluation of the control algorithms is given in Section 7.3.

Chapter 7

Experimental Evaluation

Contents

7.1 Experimental MapReduce Environment 74

7.1.1 Overview of experimental setup 74

7.1.2 Sensors and Actuators 76

7.1.3 Workload mix 77

7.2 Model experimental validation 78

7.2.1 Performance model validation 78

7.2.2 Availability model validation 81

7.2.3 Performance and availability model validation 81

7.3 Control experimental validation 85

7.3.1 Control validation scenarios 85

7.3.2 Relaxed performance - minimal resource control 86

7.3.3 Strict performance - Feedforward Control 88

7.3.4 Constrained optimal control: MR-control 94

7.4 Summary 97

In this chapter we validate the developed MapReduce dynamic models together with our on-line performance and availability control algorithms. First the experimental setup that we have built, introduced in Section 4.2, is presented in detail in Section 7.1. Second, using this setup in Section 7.2, we experimentally validate the models developed in Chapter 5. Finally, the different control algorithms elaborated in Chapter 6 are evaluated numerically and experimentally in Section 7.3.

7.1 Experimental MapReduce Environment

7.1.1 Overview of experimental setup

All the experiments in this paper were conducted on-line, in Grid5000, on a single cluster of 60 nodes, located in Nancy, France. The 60 nodes infrastructure was chosen for practical reasons, as we don't yet have access to a larger cluster size. However, all algorithms presented scale well, and can be applied to any cluster size, with only the re-identification of the equation parameters as described in Section 5.6.

Grid5000 is a French nation-wide cluster infrastructure made up of a 5000 CPUs, developed to aid parallel computing research. It provides a scientific tool for running large scale distributed experiments [14]. Each node from the cluster, used for our experiments, has a quad-core Intel CPU of 2.53GHz, an internal RAM memory of 15GB, 298GB disk space and the connection between the nodes is assured with an Infiniband 20G network.

For our experiments we use the open source MapReduce implementation framework Apache Hadoop v1.1.2 [31] and the high level MRBS benchmarking suite. A simplified version of our experimental setup is sketched in Figure 7.1. We measure from the cluster the response time, availability and the workload size. We use the cluster size and the max clients level to ensure the performance and availability objectives, regardless of the workload variations.

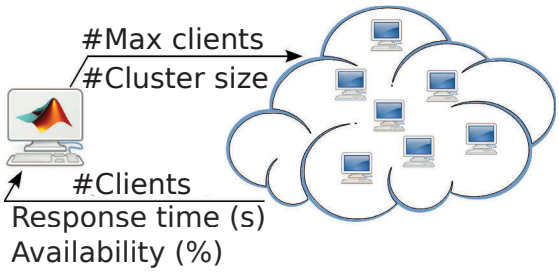


Figure 7.1: Intuitive view of the experimental setup

All the controllers were developed and implemented in Matlab. Our choice for Matlab as control environment has two main motivations. On one hand, it provides significant tools for the fast implementation and testing of different control architectures. One the other hand, it is the high level programming environment with which most control scientist are accustomed to. Therefore a control interface between Matlab and the MapReduce cluster would facilitate further research into the modeling and control of MapReduce systems, without the need for detailed low level knowledge of how the MapReduce system itself is implemented. As a result, we have built the low level scripts that allow for the control and measurement cluster performance and availability through simple Matlab function calls.

Furthermore, although Matlab already provides an array of general controllers, all the controllers presented here were implemented from scratch, and were specifically designed for our set-up. We exploited Matlab's powerful tools in the following instances:

- All the simulations of the control architecture were run in Matlab Simulink, a high level graphical block programming tool for the modeling and simulation of dynamic systems. Running simulations is important for testing validity of the control architecture and can be very useful for control parameter tuning. This was important especially in our case, where the average runtime of a single experiment was around 2 hours.
- For the identification of the model parameters we used Matlab's System Identification toolbox. See Section 5.6 for more details on this.
- For the optimal control calculation we made use of the optimization function 'quadprog' contained in Matlab's Optimisation toolbox.(Section 6.3).

For a more detailed version of our experimental setup one can check Figure 7.2. The flow of data between the local controller and the remote site is assured by encrypted tunnels created using the secure shell protocol (SSH). Most of our actuators and sensors are implemented in Linux Bash scripts.

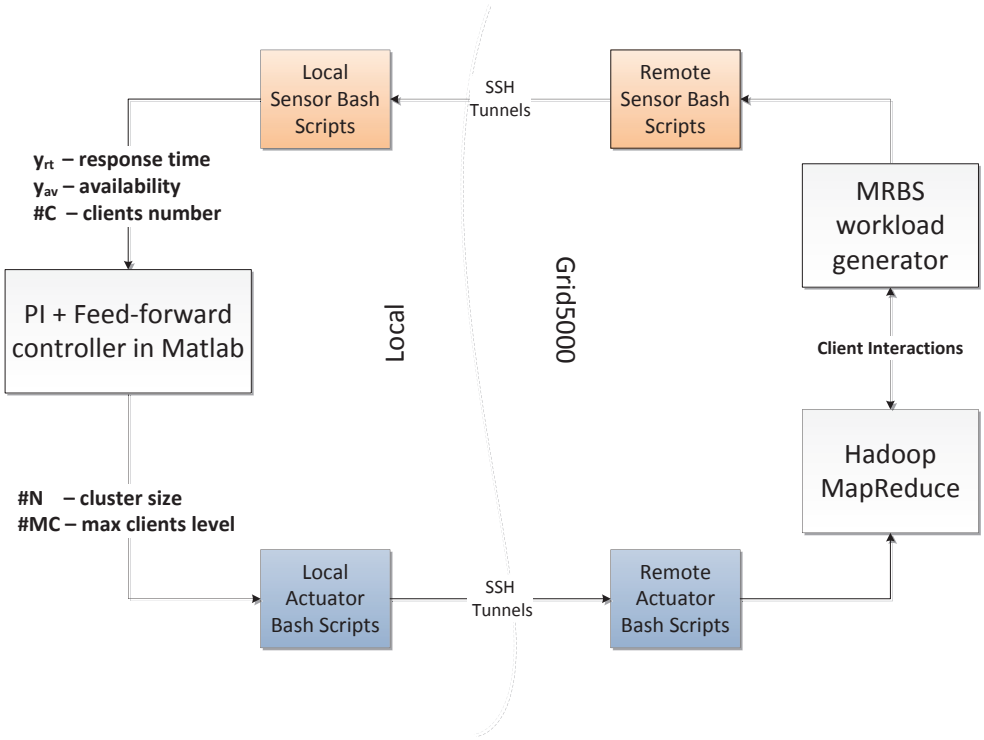


Figure 7.2: Detailed view of the experimental setup

The Secure Shell protocol (SSH) is a UNIX-based protocol, developed for the secure transmission of data over a network, between two computers. Its most frequent use is to remotely configure Web servers. SSH commands are both encrypted and secured. RSA public key cryptography is used to encrypt the data, while security is ensured by the fact that both ends of the communication channel are authenticated using a digital certificate.

A SSH tunnel is a bidirectional encrypted tunnel that uses the SSH data transfer protocol. Such a tunnel can be used to transfer unencrypted traffic between computers, over an encrypted channel. Therefore, it can be used to transfer files securely, even if the files themselves are not locally encrypted on either side.

All the experiments configuration is done locally, directly in a Matlab configuration file. Furthermore, the experiment is started from Matlab and runs completely autonomously. Moreover, the experiment data is received on-line from the remote cluster and plotted. A simplified flow of an automatic experiment run is the following:

Step 1. At the beginning of the experiment, an SSH tunnel is created from the local computer to the remote node that runs the MRBS benchmark tool. This tunnel is then used to start up the experiment using the parameters of the local configuration file (parameters such as experiment runtime, initial cluster and workload sizes). The workload size variation over time for the duration of the experiment is specified at this time and given as an input the experiment.

Step 2. As soon as the experiments starts up and the JobTracker is running, a second SSH tunnel is created to the Hadoop Master node.

Step 3. The local and remote sensor scripts, that periodically retrieve the performance and availability metrics from the MRBS runtime logs, are started.

Step 4. The on-line local plotting of the performance and availability metrics, together with the control inputs is stared.

Step 5. The on-line control loop is run for the duration of the experiment.

Step 6. When the experiment finishes the data is saved.

7.1.2 Sensors and Actuators

Our sensors and actuators are written in Unix Bash scripts. Bash scripts are shell scripts which are widely used in the UNIX world. They are excellent for speeding up repetitive tasks and simplifying complex execution logic. They can be as simple as the grouping of a set of commands, or they can perform complex tasks.

The sensor scripts run alongside the MRBS server and periodically process the Hadoop log files. This period is the sampling period T_s of the control loop. Therefore, at each time instant k $(k \cdot T_s)$ the scripts process all the log data from the last time window $(k - T)$ to

get the performance and dependability metrics. The sensor scripts are in fact a combination of bash programming and powerful Unix text processing languages such as AWK and SED. AWK is typically used as a data extraction and reporting tool, while SED (stream editor) is a compact programming language used to parse and transforms texts.

Two actuator scripts were written, one for each control input:

- One controls the cluster size N and takes as input the number nodes to add or remove to the cluster. In case of node addition, BASH scripts are written to start up the slave node services, such as the task tracker and datanode processes, for each new node that we want to add. After the processes have started up, the list of slave nodes at the master is updated with the hostnames of the new nodes. In case of node removal Hadoops command line exclude mechanism is used that safely (without any data loss) removes the nodes from the cluster.
- The other is a Java client application that connects to a RMI server, run by MRBS, to control the max clients level MC.

7.1.3 Workload mix

For our experiment we have selected the data intensive business intelligence benchmark of MRBS, consisting of a decision support system for a wholesale supplier. Requests are typical business queries and concurrent data modifications over a large amount of data (10GB). The workload used by MRBS has been taken from the decision support benchmark provided by the Transaction Processing Council (TPC). The TPC is a non-profit corporation founded to define transaction processing and database benchmarks, that deliver trusted results for the industry.

To generate the client interactions Apache Hive is deployed on top of Hadoop. Hive provides a mechanism to project structure onto our data and query the data using a SQL-like language called HiveQL. Basically, Hive converts the traditional SQL like queries into a series of MapReduce jobs.

7.2 Model experimental validation

7.2.1 Performance model validation

The identification procedure from Section 5.6 is used to find the model of the MapReduce System, noted $Y_{rt}(z)$. As given in Equation (5.1) (p. 44), Y_{rt} is composed of sum of two discrete transfer functions, $Y_N(z)$ and $Y_C(z)$. In the following, the identification results for finding the parameters of these two functions, using the prediction error estimation algorithm, are presented. Each identification experiment is run at least 3 times and the results are merged together to decrease the measurement noise.

7.2.1.1 Identifying the disturbance free performance model

The fit of the identified model for cluster size variations can be seen in Figure 7.3.

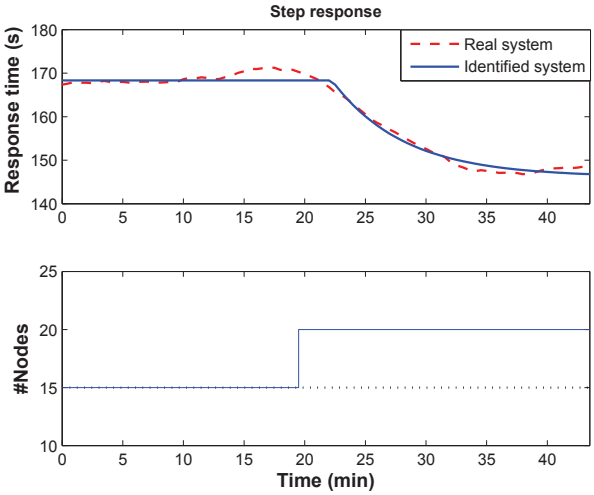


Figure 7.3: Identification of the undisturbed system. It predicts the effects of cluster size reconfigurations on performance

A step in the number of nodes is used to identify the model between the response time and the number of nodes. As it can be seen, the model found by the algorithm captures well the system dynamics with a fit level of 86.53%.

The identified form of the discrete transfer function $Y_N(z)$ is given in Equation (7.1). To better highlight the effect of each model component, the equivalent difference equation is given in Equation (7.2). This is calculated using the Z transformation properties, given in (Section 2.7.4). Using this form, we can easily see that the predicted output is calculated based on the previous output $y_N(k-1)$ value and the previous changes in the cluster size, taking into consideration the actuation delay $\tau_{rt_n} = 5$.

$$Y_N(z) = z^{-5}\frac{-0.17951(z+1)}{z-0.919} \tag{7.1}$$

$$y_N(k) = 0.919 \cdot y_N(k-1) - 0.179 \cdot N(k-5) - 0.179 \cdot N(k-6) \tag{7.2}$$

7.2.1.2 Identifying the disturbance model

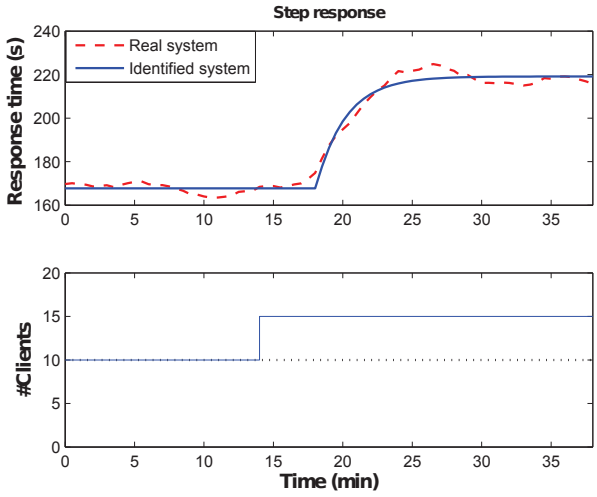


Figure 7.4: Identification of the disturbance model. It captures the effect of workload size variations on performance

Figure 7.4 shows the step responses for the identified and measured systems, in the case of changes in the number of clients. As we can see, the identified model also follows closely the measurements taken from the real system, presenting a 87.94% fit. Similarly to the previous case, the identified form of discrete time transfer function $Y_C(z)$ is given in Equation (7.4) and the difference equation form in Equation (7.4):

$$Y_C(z) = z^{-8}\frac{1.0716(z+1)}{z-0.7915} \tag{7.3}$$

$$y_C(k) = 0.7915 \cdot y_C(k-1) + 1.0716 \cdot C(k-8) + 1.0716 \cdot C(k-9) \tag{7.4}$$

One thing to notice is that for this model, the deadtime is different than the previous one, here $\tau_{rt_c} = 8$.

7.2.1.3 MapReduce Performance model

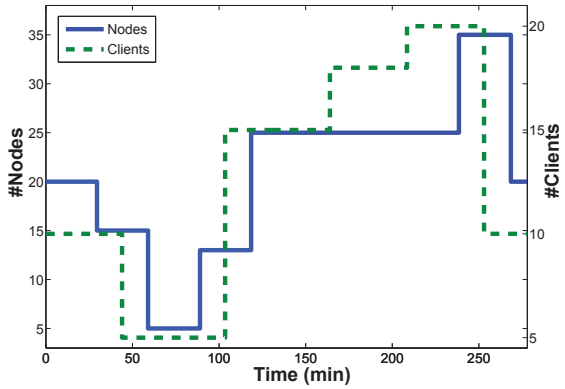
If we combine Equation (7.1) and Equation (7.3), we can write the identified MapReduce dynamic performance model, defined in Equation (5.2) (p. 45) in the modelling section:

$$Y_{rt}(z) = z^{-8}\frac{1.0716(z+1)}{z-0.7915} \cdot C - z^{-5}\frac{0.17951(z+1)}{z-0.919} \cdot N \tag{7.5}$$

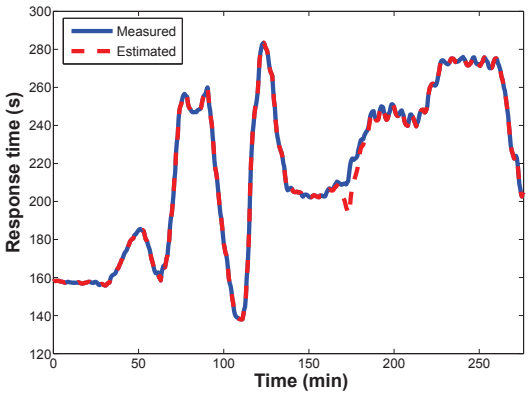
Equation (7.5) shows that both of identified discrete time transfer functions are stable, first-order systems with their poles inside the unit circle. Therefore the open loop system is inherently stable. For more information regarding system stability see Section 2.7.8.

7.2.1.4 On-line parameter adaptation

To address the non-linearities in response to cluster scaling, that appear when the system leaves the operating region, the model parameters can be adapted on-line. The experiment in Figure 7.5 shows the accuracy of the adaptive model in capturing the system response to workload and cluster changes. We can see the model estimations follow closely the measurements taken from the real system. The parameter adaptation technique used in this case is the recursive least square estimator described in Section 5.6.2. For more details one can check the master thesis of Sophie Cerf [17].



(a) Inputs - workload and cluster sizes



(b) Performance

Figure 7.5: Performance model validation [17]. It predicts the effects of cluster and workload size variation on performance

7.2.2 Availability model validation

The fit of the identified model for max clients level variations can be seen in Figure 7.6.

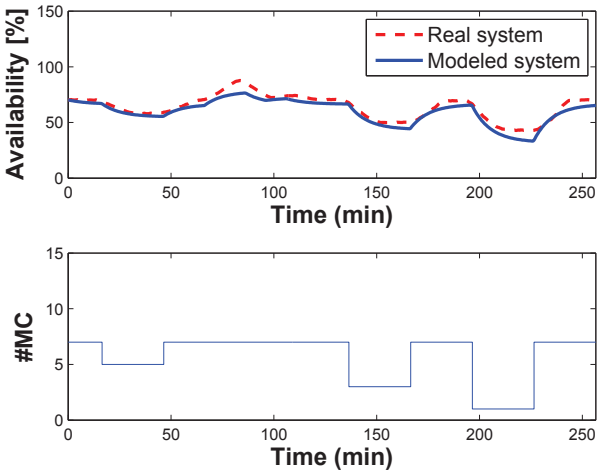


Figure 7.6: Availability model validation with $N = 20, C = 10$. It predicts the effects of max client level variations on availability

The figure shows responses for the identified and measured systems, in response to MC variation. The identified model follows closely the measurements taken from the real system. The identified form of $Y_{av}(z)$ is given as a discrete time transfer function in Equation (7.6) and in a difference equation form in Equation (7.7):

$$Y_{av}(z) = -\frac{0.1548}{z - 0.946} \cdot (C - MC) \tag{7.6}$$

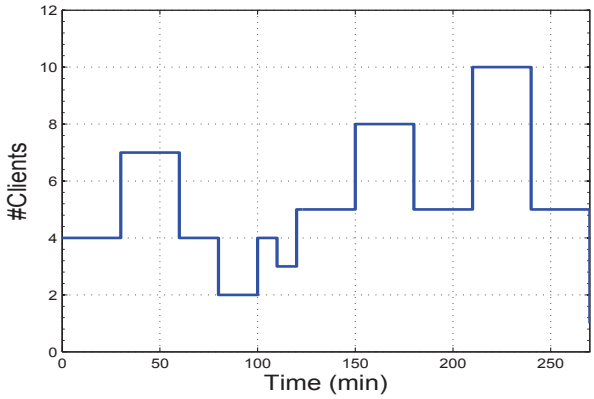
$$y_{av}(k) = 0.946 \cdot y_{av}(k-1) - 0.1548 \cdot C(k) + 0.1548 \cdot MC(k) \tag{7.7}$$

The identified difference equation shows that the effect of MC variations can be calculated based on the previous output $y_{av}(k-1)$ value and the current MC and C values.

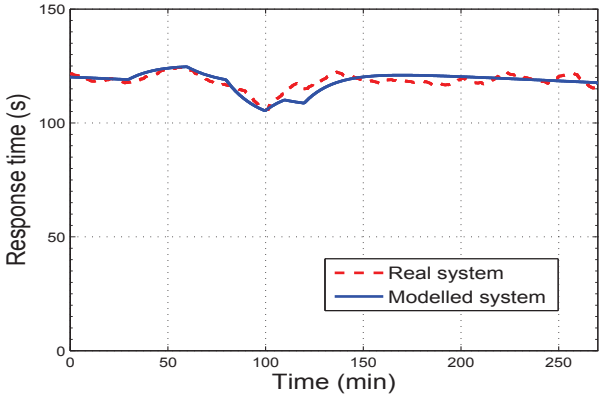
7.2.3 Performance and availability model validation

To validate the accuracy of the MIMO performance and availability model, proposed in Section 5.5, two type of validation experiments are presented. These were designed to evaluate the model's ability to capture the effect of cluster size and workload variation on the system performance and availability.
In the first case we fix $MC = 5$, $N = 20$ and the workload size is varied between 2 and 10 clients, see Figure 7.7. The upper two graphs show the evolution over time of the performance and availability for both the real system (dashed) and the modelled one (solid line). The

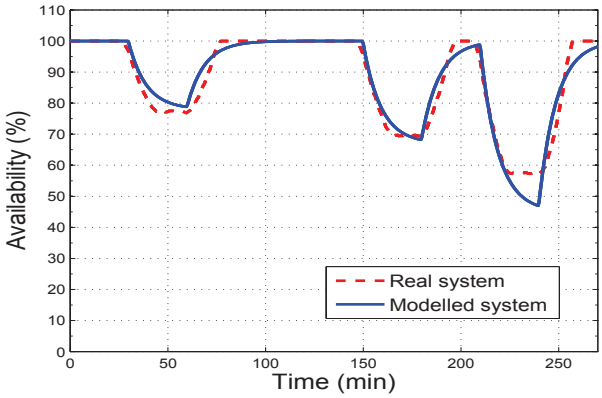
results show that the model accurately reflects the behaviour of the real system. For instance we can observe that any changes that are under the $MC = 5$have an effect upon the response time but not on availability. Meanwhile, when the number of clients exceeds MC, availability is affected and response time remains unchanged.



(a) Clients

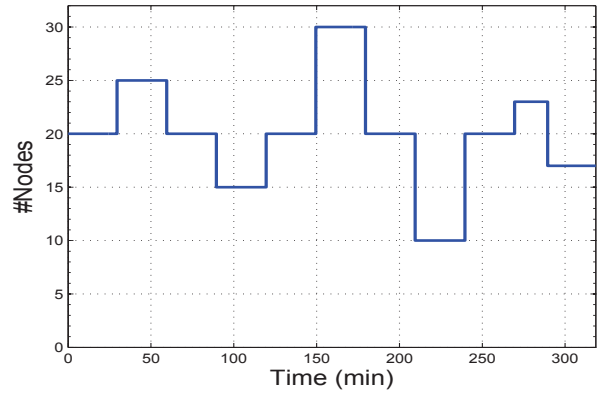


(b) Performance

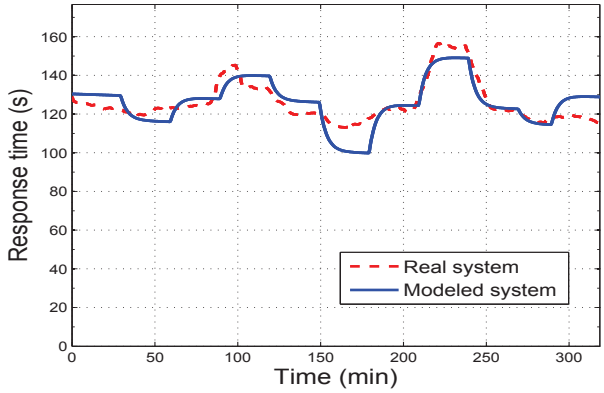


(c) Availability

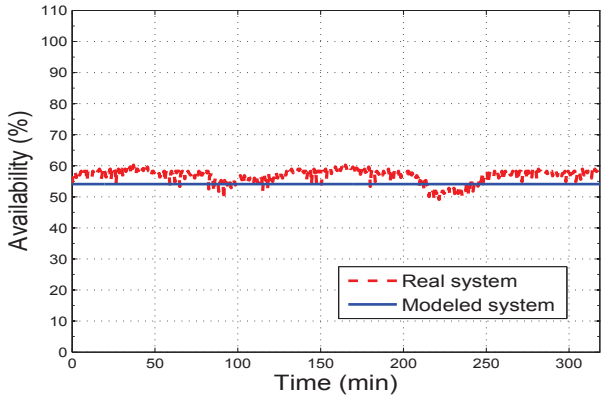
Figure 7.7: MIMO model validation for workload variation, #MC=5, #Nodes=20



(a) Cluster size



(b) Performance



(c) Availability

Figure 7.8: MIMO model validation for cluster size variation, #MC=5, #Clients=10

In the second round of experiments we fix $C = 10$ and $MC = 5$ and we vary the cluster size from 10 to 30 nodes, see Figure 7.8. Although the model captures well the effect of these nodes changes, we can see small differences coming from the fact that some non-linearities

arise when adding (removing) a large number of nodes at the same time. However as we will show further on, the model accuracy is sufficient for our control algorithm, since the feedback optimal controller can compensate for such small non-linearities. The effects of MC variation has been already plotted in the previous section in Figure 7.6. Here, we just want to note that it's model has the highest accuracy out of the three inputs.

To conclude, the model validation experiments prove that our models can successfully captures the dynamics of the real system.

7.3 Control experimental validation

7.3.1 Control validation scenarios

In this section we validate our control architectures developed in Section 6.1 based on different use case scenarios that are relevant for cloud systems. The primary focus of these scenarios is to test the control response to the practical, real demands of any cloud controller architecture.

First, for the case when we are interested in controlling only the performance of MapReduce systems, we identified two large industrial use cases:

1. First, we have the relaxed performance - minimal resources case where the service provider needs to keep the system response time below the reference threshold defined in the SLA, but also wants to minimize the number of resources it utilizes (in this case the number of system nodes), to reduce cost. Therefore, if this is specified in the SLA, the client accepts that *for a small amount of time* the response time could exceed the reference threshold.

2. The second case is the strict performance one, when the service provider has a *very strict demand* from the client in keeping the response time below the reference, defined in the SLA, all the time. This can be the case for online brokerage industry where the service unavailability costs about 6.48 million dollars per hour [23]. Since the number of clients trying to use the service is unpredictable, the service provider is accepting a considerable increase in the number of the system nodes (therefore an increase in the utilization cost) in order to respect the SLA and face the client increase.

Second, in the event that the controllers have multiple concurrent objectives, such as performance and availability in our case, two scenarios were designed in order to highlight the situations when the trade-off between the two objectives needs to be addressed by the controller. Of course, when we have no limitations on resources and the outputs are achievable, the controller will keep both availability and performance objectives. However, in case of strict input or output constraints, the controller needs to enforce one of the outputs at the expense of the other one.
To highlight how our control architecture tackles the latter conflicting objectives we developed the following two control use cases:

1. *Performance guardian* guarantees response time, while maximising availability and minimising cost.

2. *Availability guardian* ensures availability, while minimising response time and cost.

In the following, these scenarios are examined in both simulations and experiments with the real MapReduce cluster described in Section 7.1.

7.3.2 Relaxed performance - minimal resource control

In this section the classical and event-based feedfoward control architectures, designed for the Relaxed performance - minimal resource control use case, are experimentally evaluated.

7.3.2.1 Classical feedback control

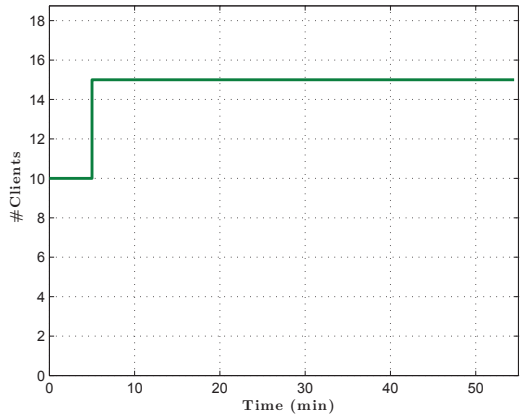
First, we consider a simplified version of the control architecture from Figure 6.1 (p. 56), where $U_{ff}(z) = 0, \forall z$. Therefore, the behaviour of the classical PI feedback controller $U_{PI}(z)$, developed in Section 6.1.2, is examined.

The equation of the sampled time PI controller utilised has been given in Equation (6.2) (p. 58). The parameters of the controller are determined through loop shaping, to assure closed loop stability and no control overshoot. In control theory we talk of output overshoot when the system output exceeds the reference value before stabilizing on it. However, in the case of cluster scaling the control profile can be just as or even more important then the output. Therefore, as we would like to avoid a highly aggressive controller, that adds and then removes nodes in short amount of time, the controllers response to the disturbance is designed to be slow. By doing this, we minimise the number of the cluster reconfiguration instances through decreasing the control reactivity, which in turn decreases the control overshoot. Still, one important thing to keep in mind is that the more we decrease control reactivity the larger the overshot will be in the output because of the slow control response.

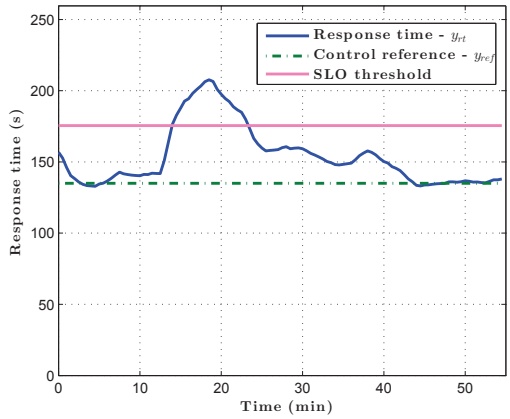
Taking all these requirements into account, we computed the value of $K_p = 0.0012372$ and $K_i = 0.25584$ for our controller. The results are given in Figure 7.9, which shows our controllers response to a 50% change in the number of clients (Figure 7.9a). We can see that, as the controller is determined to have a slow settling time, the SLA threshold is breached for a short amount of time. Nevertheless, the controller takes the response time to the reference value, by steadily increasing the number of nodes until response time recovers (Figure 7.9d). The cluster size required to keep the SLA is recomputed at each new sampling interval (Figure 7.9c). Therefore, care must be taken to avoid a reactive control configuration and to minimise measurement noise levels, otherwise we can have frequent cluster resizing.

Notice, that in this case the reference value is not set exactly to the SLO reference threshold. The reasoning behind this is that, in traditional control theory the control objective is to reach an exact reference value. Traditional cloud SLO objectives such as keeping a signal value below a certain threshold can not be directly formulated, in this setup. Therefore, in case of feedback control, to give time for the controller to react this reference value it should always be a value close to, but below the SLO threshold.

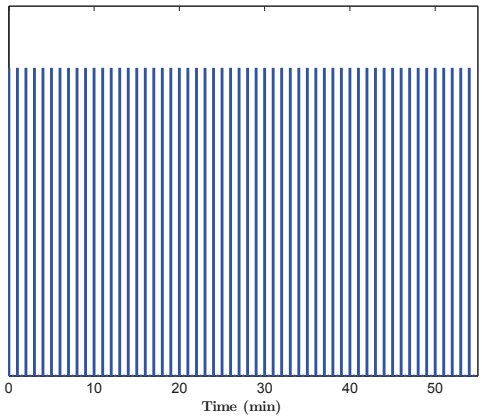
The main advantage of this type of control is its simplicity and high robustness to modeling errors as well as unknown disturbances. This is a key factor for any cloud controller as there are many points of contention that can arise in such complex system as MapReduce.



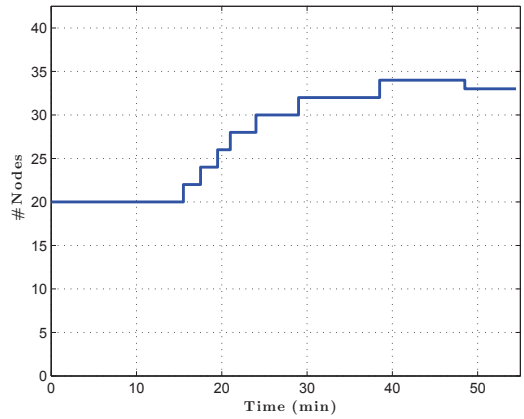
(a) Workload



(b) Performance



(c) Control calculation instants



(d) Cluster size

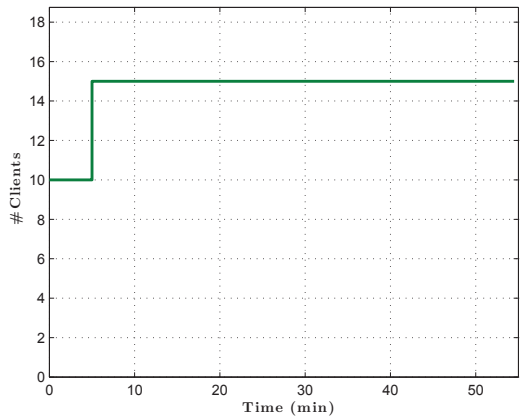
Figure 7.9: Classical PI feedback control - experimental evaluation

7.3.2.2 Event-based feedback control

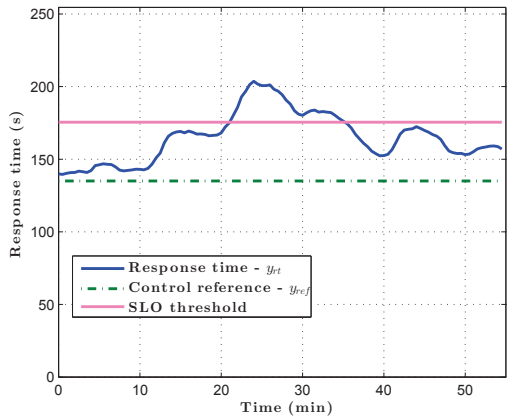
To highlight the benefits of even-based feedback control, developed in Section 6.2.2, in comparison to the previous classical one, we use the same controller configuration parameters, $K_p = 0.0012372$ and $K_i = 0.25584$, and same control scenario as in Section 7.3.2.1.

Figure 7.10 shows that the event-based PI controller, with detection level $\bar{e}_{fb} = 5$, also manages to recover the response time and keep it below the threshold in the presence of perturbations. One can also see that, when comparing Figure 7.9c with Figure 7.10c, the control to be applied is computed much less frequently. Furthermore, if we compare Figure 7.9d with Figure 7.10d, we can see that the number of cluster reconfigurations is drastically diminished as well. In fact the number of cluster reconfigurations is decreased by half (4 changes with the event-based controller compared to 8, without). Although the performance is somewhat worse, we do have significant gains in terms of a smoother control profile and less control

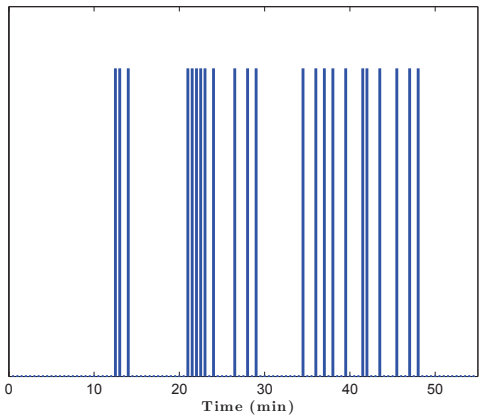
reactivity. Which in turn brings less energetic cost for the cloud provider and less financial cost for the user.



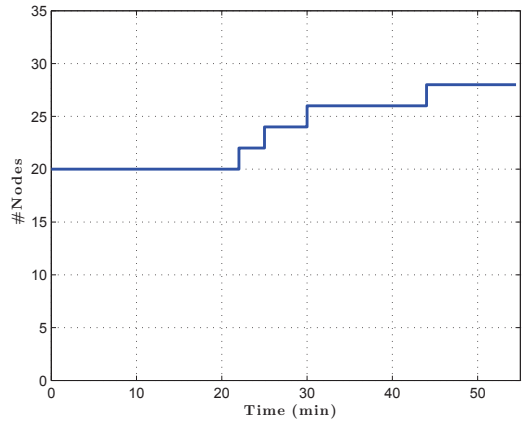
(a) Workload



(b) Performance



(c) Control calculation instants



(d) Cluster size

Figure 7.10: Event-based PI feedback control - experimental evaluation

7.3.3 Strict performance - Feedforward Control

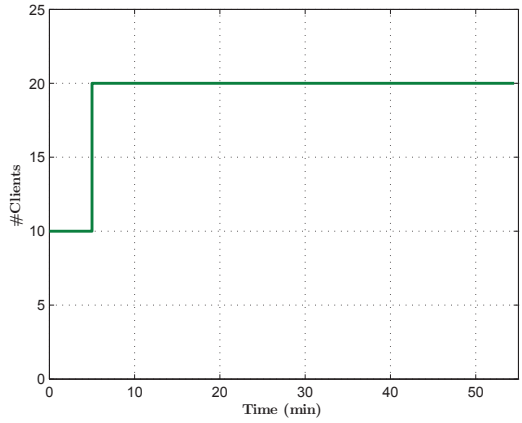
In this section the classical and event-based feedforward control architectures, designed for the strict performance use case, are presented.

7.3.3.1 Classical feedforward control

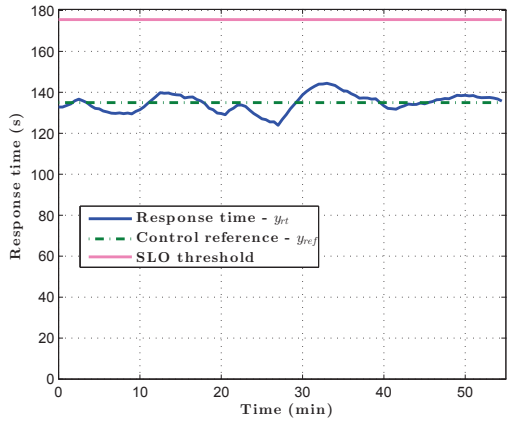
The effect of adding the feedforward controller $U_{ff}(z)$, given in Equation (6.3) (p. 59), to the previous PI feedback control, can be seen in Figure 7.11. The implemented difference equation

form of the computed feedforward controller is given in Equation (7.8):

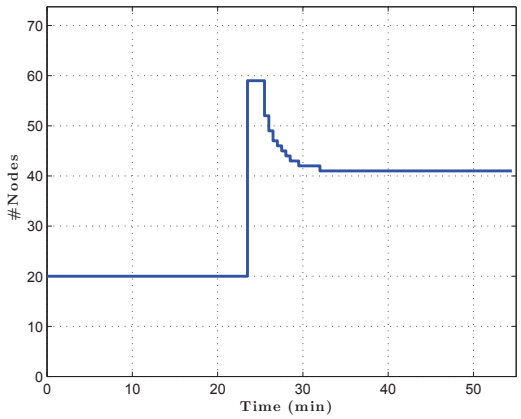
$$u_{ff_{(k)}} = 0.791 \cdot u_{ff_{(k-1)}} + 5.9698 \cdot C_{(k-2)} - 5.486 \cdot C_{(k-3)} \tag{7.8}$$



(a) Workload



(b) Performance



(c) Cluster size

Figure 7.11: Classical feedfoward control - experimental evaluation

One can also observe that, although so far we have tested our controllers with a jump of 50% more clients, here we test our control with a disturbance twice as worse. We have a jump of 100% more clients, to highlight the effectiveness of feedforward control in comparison to just feedback control. While the feedback control shortly breached the SLO even with a 50% increase, by adding the feedforward component we can see that the controller response is increased and manages to keep the response time below the SLA threshold at all time, even if we double the current workload. This comes though with an increase in control cost, due to the larger cluster size and the control profile utilized. Being a time based controller, the number of nodes to be added is recomputed at each sampling interval and therefore it is not

plotted again. Furthermore, the feedback term compensates for all the model uncertainties that were not considered when calculating the feedforward and assures that the steady state error converges to 0.

The reason why feed-forward control does so much better then just feedback control is the large delays in response to a disturbance and to an actuation, that we have in such systems. For example in our case we have an 8 sampling period delay (240 second) from the time the disturbance appears (clients arrive), until their effect can be measured on the output. Furthermore we have a 5 sampling period delay (150 second) in the actuation, namely the time it takes for the effect cluster scaling to be measurable on the output. Meanwhile, in cases such as ours, we can measure immediately when the new clients arrive and we can use this information straight away to intervene, instead of waiting for seeing the effects of the disturbance on the output, as it is in the feedback control case. This is where the strength of feedforward control comes into play. We can instantly counteract through cluster scaling (Figure 7.11c), the effect of the workload variations (Figure 7.11a) at the same time as they are affecting the system and practically nullify the disturbance effect on the output (Figure 7.11b).

7.3.3.2 Event-based feedforward control

In this section we evaluate the event-based feedforward control architecture introduced in Section 6.2.3.

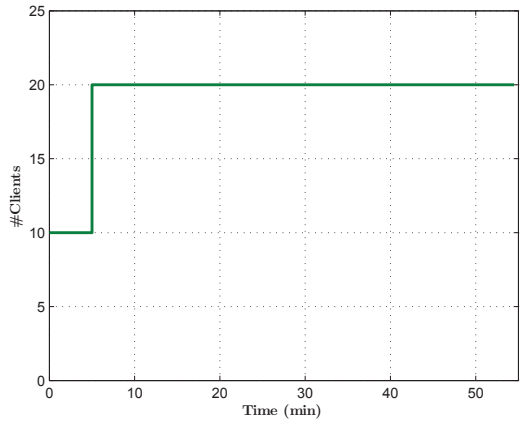
For the first validation experiment we design an event function to reduce the number of cluster reconfigurations. The chosen event signal, shown in Equation (7.9) is a virtual one, which is the predicted output of the disturbance model, given in Equation (7.4), in response to the measured workload variation C.

$$\epsilon_{ff}(t) := \bar{e}_{ff} - U_{ff} \cdot \big|(C(t) - C(t_j)\big| \tag{7.9}$$

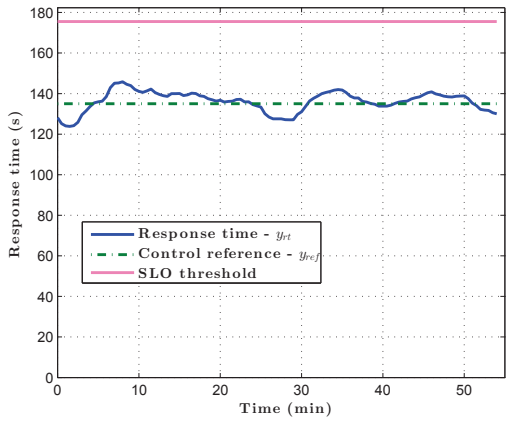
where the event detection level is $\bar{e}_{ff} = 10$ and U_{ff} is given in Equation (7.8). Figure 7.12 shows the effect of adding the event-based feedforward controller on performance. Similarly to the event-based feedback controller, we can see the number of cluster reconfigurations is reduced by half compared to the classical case presented in Figure 7.11, meanwhile we still benefit of a good disturbance rejection, keeping the response time stable.

Additionally, as we can see the event function can be used to minimise the node removal frequency, introducing a sort of a delay between node removals. This delay in taking control actions is important because, we have to remember that in case of such systems as MapReduce we can have the multiple other optimisation functions (schedulers), besides our controller, that optimise the cluster continuously, therefore a succession of quick reconfigurations is not desirable in practice.

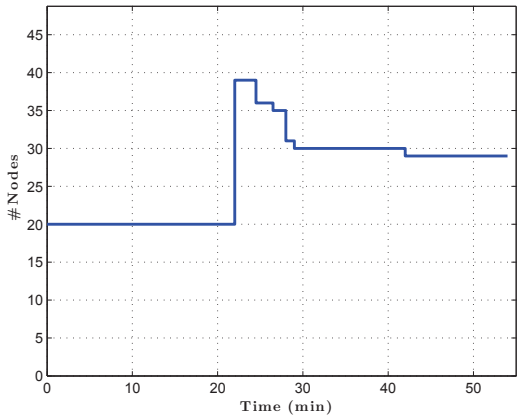
Note also that, whereas the feedback controller U_{PI} is not required in the ideal case, here it is compulsory to ensure the stability and robustness of the closed-loop system. See Section 2.7.8 for the intuitive definitions of system stability and robustness.



(a) Workload



(b) Performance



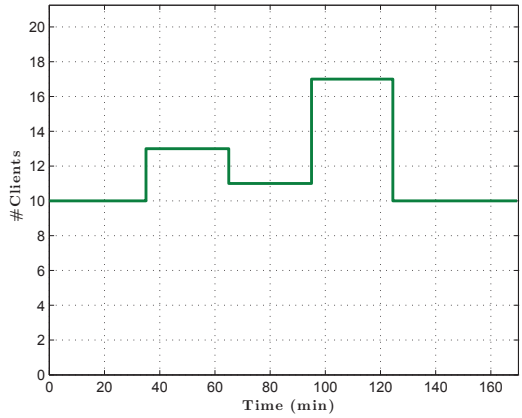
(c) Cluster size

Figure 7.12: Event-based feedforward control - experimental evaluation

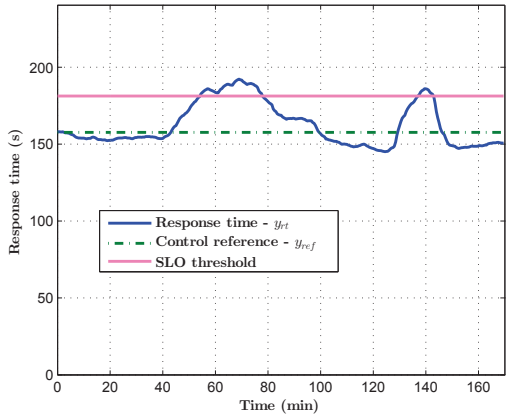
Nevertheless, if we take a look at the feedforward controller in Figure 7.12 we can see that, its behaviour is to initially add a larger number of nodes and than to slowly decrease down to the new stabilizing level. This behaviour comes from the dynamics of the disturbance effect. Its purpose is to enact through cluster scaling the inverse effect that nullifies the disturbance influence on performance. While, this approach compensate exactly for the changes in the system, such a behaviour should be used only when we have strict-performance requirements, when keeping performance is much more important than the control cost. In cases when both cost and performance are important concerns, the natural question arises: how could we use the benefits of the feedforward controller, without the previously described control profile?

We have provided the answer to this question in Equation (6.13) (p. 66), which is a modified version of the traditional feedforward controller, that doesn't make use of the complete disturbance dynamic. Instead it uses only the feedforward gain, basically transforming the feedforwad controller into a simple proportional controller. The gain of this proportional

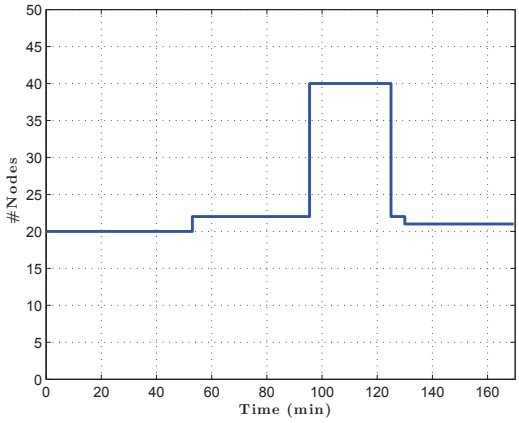
controller is calculated through the division of the disturbance model gain with the process gain (Equation (6.13)). The event signal in this case is directly the workload size variation C and the event function defined in Equation (6.14). By defining the event function directly on the workload lets us to better combine the strengths of the feedback and feedforward control architectures. On the one hand, a slow event-based PI feedback controller is designed that is in charge of responding any small variations around the operating point caused by unknown disturbances, such as bottlenecks and small workload variations, meanwhile ensuring all round system robustness. On the other hand a fast but low cost event-based feedforward controller is added to handle large workload variations that are above the natural workload variance limit.



(a) Workload



(b) Performance

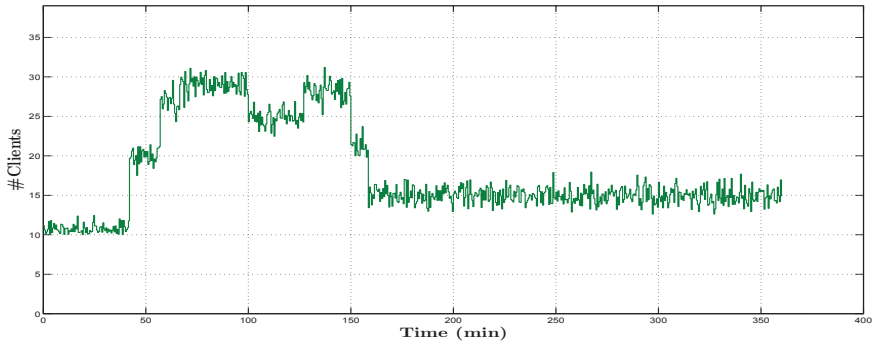


(c) Cluster size

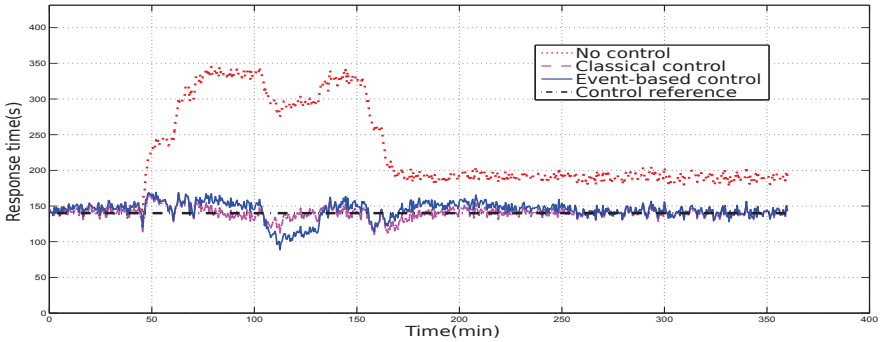
Figure 7.13: Modified event-based feedforward control - experimental evaluation

The results of such a control setup is presented in Figure 7.13. Notice that the initial small client variation, from 10 clients to 13 and then to 11, is handled by the feedback PI controller. Meanwhile, the large workload change, from the 11 to 17 is handled by the feedforward

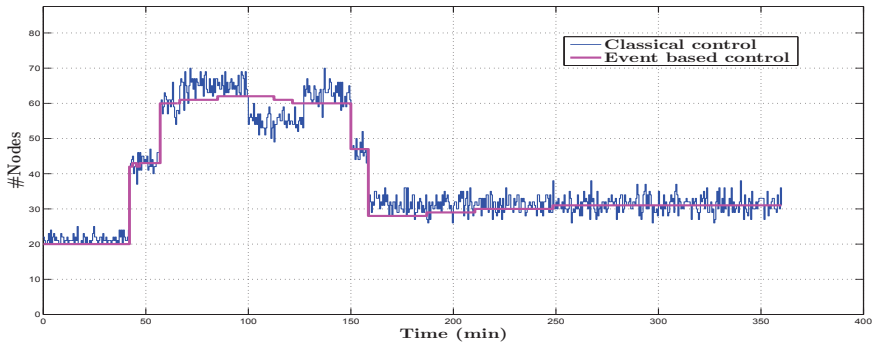
controller. If we further compare the modified control response given in Figure 7.13c with the traditional one given in Figure 7.12c we can see that in the modified case the controller adds directly the nodes required after the absorption of the initial shock. We can also5 see that although we loose a bit in terms of performance during the initial shock (Figure 7.13b), we gain a smooth control profile and drastically reduced costs (Figure 7.13c).



(a) Workload



(b) Performance



(c) Cluster size

Figure 7.14: Event-based feedback and modified feedforward control - evaluation in simulation

In Figure 7.13 we can see that the control profile is much improved compared to the

classical case. To highlight even better the benefits of the modified feedforward controller let us look at the following simulation of a more complex, highly varying workload (Figure 7.14a) in Figure 7.14, where we compare the classical and event-based control performances.

When comparing the two control profiles of Figure 7.14c it is clear to see that the event-based controller drastically decreases the cluster reconfiguration count, having up to 30 times less reconfigurations at te end of the experience in the event-based case. Meanwhile the performance responses of the two controllers are similar (Figure 7.14b).

7.3.4 Constrained optimal control: MR-control

In this section we present the twofold evaluation of the control law *MR-Ctrl*, proposed in-Section 6.3. The evaluation scenario was chosen, so that we could highlight the two control scenarios, defined in Section 7.3.1. Therefore, we limit the maximum cluster size to be 40 nodes and increase the workload size sufficiently, so that the controller can't ensure both outputs objectives simultaneously.

We start of with $N = 20$, $MC = 9$ and $C = 10$. The duration of the evaluation experiments is 130 minutes. During the experiment we vary the workload twice: we have 4 new clients that arrive at 30 minutes and then 5 clients leave at 70 minutes, see Figure 7.15a.

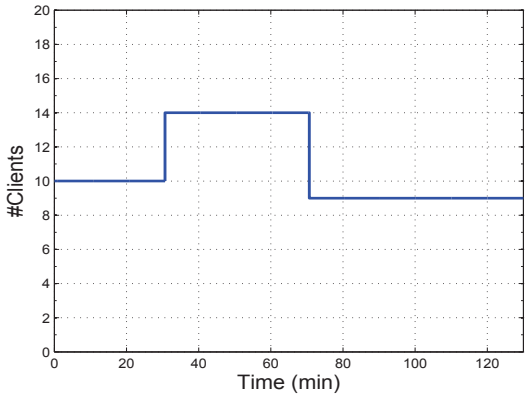
The control objectives, for all the evaluation experiments, are fixed as $Av > 90\%$ and $Rt < 125s$.

First we validate the control strategy in simulation. These simulations help in fast and inexpensive testing, tuning of the controller in different scenarios. The closed loop response of the MapReduce system with both controllers can be seen in Figure 7.15.

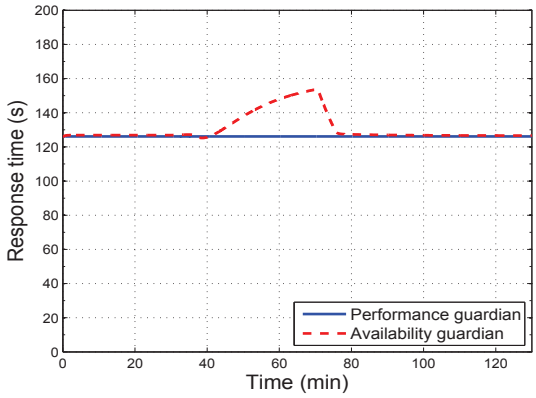
In the case of the *Performance guardian*, the controller optimally counter effects the workload changes and keeps a stable response time, while maximising availability and minimising resource usage. If we take a closer look at the controller responses, such as the cluster size (Figure 7.15d) and MC (Figure 7.15e) variation, we can see that when the controller reaches the upper nodes limit of 40, it decreases the MC to keep the reference response time. This in turn leads to a decrease in system availability, see Figure 7.15c.

In the second case of the *Availability guardian*, ensuring availability is preferred to performance. Therefore the controller first ensures the desired availability levels, while at the same time minimises response time and resource usage. If we compare the control inputs to the previous case, Figures 7.15d and 7.15e, we can see that in this case the controller keeps MC levels stable to guarantee availability and uses the maximum amount of nodes available to minimise response time.

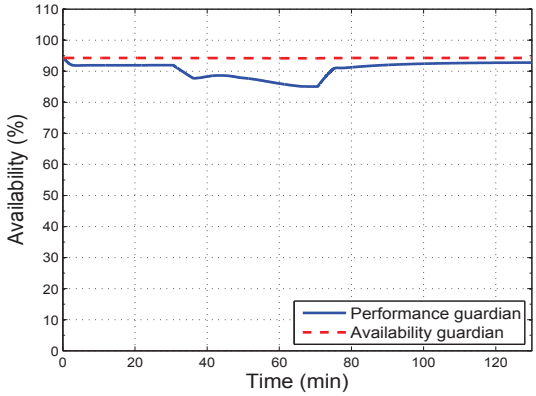
Now we test our control algorithm using the MRBS benchmark tool and a Hadoop MapReduce system running in Grid5000. We use the same workload scenario as in the simulations.



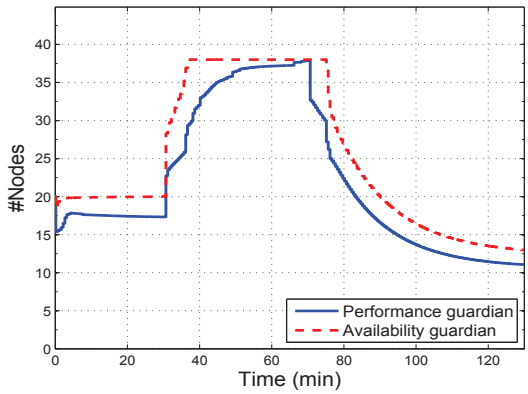
(a) Clients



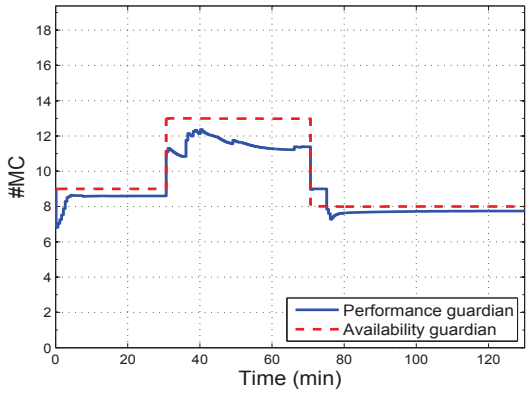
(b) Performance



(c) Availabilty



(d) Cluster size

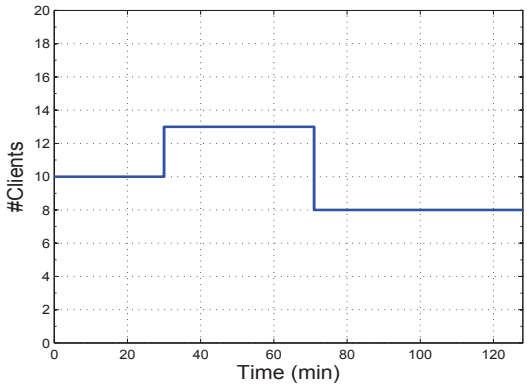


(e) MC

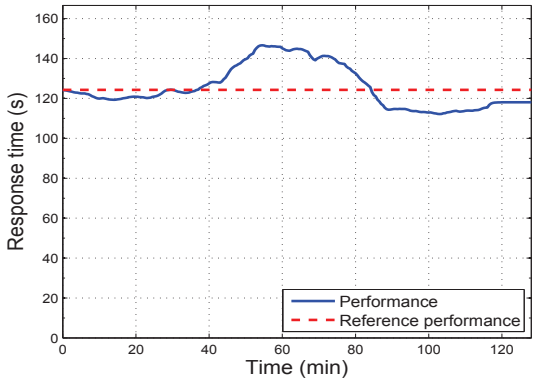
Figure 7.15: Performance guardian and Availability guardian control scenarios - evaluation in simulation

Because the real system has a natural output variance of around 5%, we add a safety band of 5%, around our performance and availability objectives, where no control is calculated.

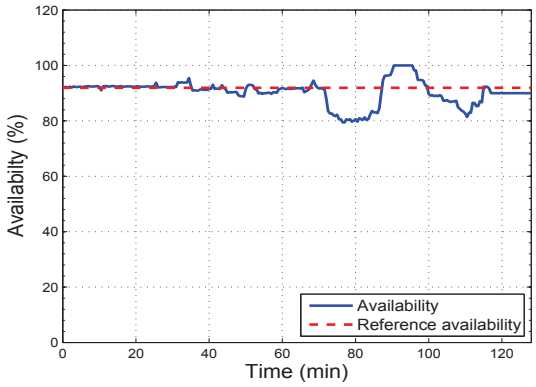
Furthermore, as in real life scenarios one would want to avoid the continuous cluster size reconfigurations, using the weighting factor R, see Equation (6.16) p. 67, we put a penalty on cluster scaling operations.



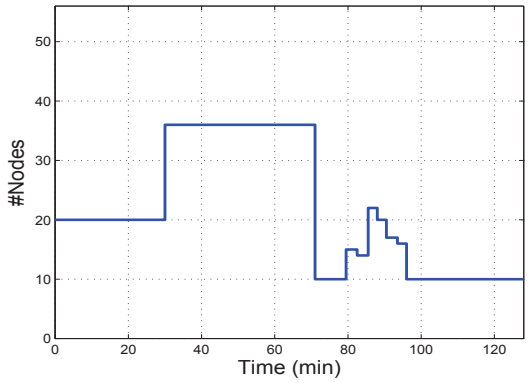
(a) Clients



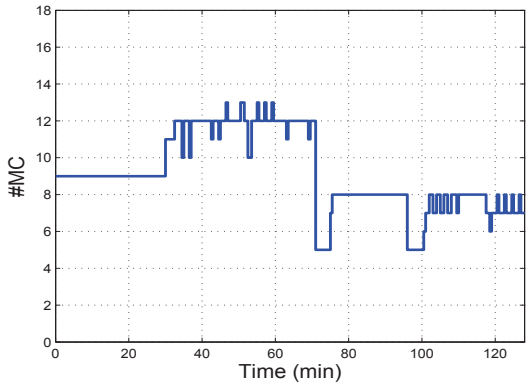
(b) Performance



(c) Availability



(d) Cluster size



(e) MC

Figure 7.16: Availability guardian control scenario - experimental evaluation

To prove the effectiveness of the control algorithms on the real system an experiment is shown using the Availability guardian configuration. Figures 7.16a and 7.16b show the response time and availability variations of the real system. We can see that at the beginning of the experiment the controller manages to keep the availability objectives, sacrificing performance to do it. After the second workload change, when the number of clients decreases, both availability and performance objectives are met. We can also observe small overshoots in terms of response time and availability, which may come from some unmodeled external disturbance factors that influence real MapReduce deployments. In Figure 7.16d we can follow the effect of the penalty on node changes. There are much less cluster size reconfigurations, than in the simulations presented before, where we didn't have this penalty. Moreover, MC level is varied more freely since there is no cost associated to changes in MC, Figure 7.16e.

7.4 Summary

In this chapter we present the successful experimental validation of the developed models and on-line control algorithms. First, in Section 7.1 we present the experimental setup that makes it possible to test different on-line control algorithms on a real MapReduce cluster, directly from the comfort of the Matlab programming environment. Second, in Section 7.2 we show that the models developed in Chapter 5 can successfully capture the performance and dependability of a MapReduce cluster. Finally, in Section 7.3 the efficacy of the different control algorithms elaborated in Chapter 6 is highlighted, both numerically and experimentally, in varied cloud scenarios.

Chapter 8

Conclusions and Perspectives

8.1 Conclusions

The research subjects addressed in this thesis are situated at the crossroads of two communities: informatics and control. The main contributions of the thesis are the modelling and control of the performance and dependability of MapReduce cloud software systems. MapReduce is one of the currently most popular parallel processing algorithms. It is a complex framework that offers a rich setting, in data and workload sizes and types, for the development of modelling and control techniques of highly data intensive applications, specific to our BigData age. In the following paragraphs the **thesis contributions** are summarised:

I. We have implemented an autonomous on-line control framework, that allows to measure and control the performance and availability of a MapReduce cluster from any local computer running Matlab. The control framework developed can be easily adapted to any other software system. The choice of Matlab, as the controller development environment, was made in order to facilitate further research into such software systems by the control community.

II. We propose the first algorithm for the dynamic modelling of the performance and availability of a MapReduce cluster, running a concurrent data intensive workload. Using this algorithm, several black-box performance and dependability models are proposed. All the presented models have been successfully validated on a real 60 node MapReduce cluster, in Grid5000, running a data intensive Business Intelligence workload.

   The models were found to capture the dynamic behaviour of a MapReduce system with high accuracy. Nevertheless, to improve upon model robustness and address possible system non-linearities, on-line adaptation techniques for calculating the model parameters are described as well.

III. A control architecture based on classical time-based feedback and feedfoward controllers is developed to ensure MapReduce performance through cluster scaling. The controllers were found to be successful in ensuring performance constraints. The advantages of this control approach, compared to existing solutions, are the mathematically ensured control loop stability and the robustness to modelling and environmental uncertainties.

   Still, the control profile and reactivity were found in need for further improvement and adaptation to the cloud scenario.

IV. To improve the control profile and provide more options in designing the control reactivity, an event-based feedback and feedforward control framework is developed. Moreover, a novel modified event-based feedforward approach is introduced, that retains most the benefits of the predictive behaviour of a classical feedforward control, but minimises control cost by reducing the control dynamics.

    The event-based method, by reducing control reactivity, has effectively cut by half the number of cluster reconfigurations compared to the classical feedback approach, thus diminishing considerably costs. Meanwhile, the decrease in performance is negligible. Moreover, the control profile is greatly improved by minimising undesired control behaviours, such as oscillations in the control inputs.

    Still, this approach can ensure only one control objective at a time and the optimality of the control profile is not implicit.

V. Finally, the optimal control framework *MR-Ctrl*, that can optimally ensure multiple output objectives simultaneously is developed. In our case it can optimally control at the same time both the performance and availability of a MapReduce service, meanwhile explicitly minimising the control cost as well. Moreover, whenever both objectives can not be met due to cost or resource limitations, *MR-Ctrl* can successfully handle the trade-off between performance and availability according to predefined priorities.

    The biggest advantage of the method consists in its wide applicability, as most control problems for software systems can be reduced to an optimisation problem. Furthermore, the control framework works the same regardless of the number of input-output variables and the optimal solution is found after a finite number of optimization steps. Moreover, the framework can implicitly handle actuation delays and even dynamic input limitations.

    One of the limitations of this approach is that it is based on a linear system model. However, in practice having a linear model is sufficient for most systems, especially if the model parameters are adapted on-line.

Furthermore, taking our experiences in modelling and control MapReduce system we highlight some **general conclusions** regarding our experience with the use of control theory for the modelling and control of such cloud software systems:

- One of the most important steps, before applying control theory to any software system is defining the correct output signal, one that varies over time and based upon which the control decisions need to be taken. We know that we have a good output metric if the measured output signal is stable when our inputs are not changing and if it keeps its directionality in response to input changes. All in all, selecting suitable output signals is highly important because it conditions the complexity of the control algorithms.
- When it comes to modelling, there are two main approaches to take. Either one tries to build a detailed model of the system or use more general, black-box technique that make no assumption about the inner workings of the system, but instead fits a model to the experimental data. We found that the later approach is much more suitable for software systems because, as there are no physics governing software, a detailed dynamic model

is difficult to achieve. Not to mention that any of the frequent software updates can instantly change the inner workings of the system.

Moreover, due to the large environmental uncertainties where such software run, like the cloud, static modelling is not sufficient. In our experience, even when using the same cluster and workload sizes, we sometimes had significant offsets in terms of performance. Therefore, on-line parameter adaptation techniques that periodically update our model parameters, continuous adaptation of our models to the current environmental conditions is advisable.

- Maybe somewhat counter intuitively, in control theory building simple models, that capture the general behavioural tendencies of system is preferred to complex ones, that capture all the small non-linearities of the real system. The reason for this is that the more complex the model is, the more difficult it is to build the control laws and the less general the approach becomes. Therefore, the focus is on keeping the models of the system as simple as possible and making the control law sufficiently robust to handle the modelling uncertainties and non-linearities.

- One other important issue we encountered, when modelling cloud software systems, is the large input/output delays. The understanding of how these delays affect the system and their incorporation in the modelling and control solutions is crucial. Delays can have a huge impact on the stability of the feedback control loop and their mismanagement can lead to oscillations in practice.

Finally, let us summarise the conclusions on the **practical applicability of the different control approaches** developed in this thesis. Out of the control solutions that we have developed in this thesis, the event-based architecture and the optimal control framework *MR-Ctrl* proved to be the most promising.

- The advantage of the event-based PI and feedforward framework is its simplicity in implementation . Furthermore, through the event functions, one can customise the control reactivity and profile according to the requirements and it also has an intuitive connection to the existing threshold based control solution. Moreover, there are numerous practical solutions in the speciality literature for the automatic tuning of such feedback controllers. Nevertheless, this method can be applied only for cases when we have single output objective to guarantee.

- The ability to control multiple objectives at the same time is a big advantage of *MR-Ctrl*. This is a general, optimal control approach, providing a unified treatment of systems, regardless of the number of input-outputs. Not to mention, that most control problems, in case of software systems, can be formulated as an optimisation problem. However, the implementation and tuning of the framework requires a higher level of control theoretical expertise.

8.2 Perspectives

This dissertation is the first step towards the dynamic modelling and control of cloud software solutions, taking the popular MapReduce framework as our use case. In the following we list a few interesting research directions we envision to complement and to extend upon the work presented in this thesis.

- In this thesis we use response time, as performance and respectively availability, as dependability metrics. In some cases other quality of service metrics should also be taking into consideration, such as throughput and reliability.
- Extending the time-based and event-based control frameworks with adaptive modelling and control techniques. This subject is described in more details in the Master thesis of Sophie Cerf [17].
- Investigating the viability of non-linear modelling and control solutions.
- Investigating the use of optimal control as a general reaction strategy in case of systems of systems, see the European Project AMADEOS [56].
- The use of these feedback control techniques for the adaptation of the algorithms and the internal behaviour of software systems. For more on this subject, see our systematic literature review on the subject [71].
- The testing of the proposed control solutions under different cloud environments, such as the Amazon, Google and Microsoft cloud platforms.

The proposed solutions are general enough to be applicable to a number of cloud software systems. Nevertheless, cloud systems provide highly variable environments for software services, therefore more research needs to be done toward the on-line adaptation of the parameters of both the model and the controller.

The ultimate goal is to provide a package of "off the self" modelling and control solutions for cloud performance and dependability management, which can be used by any system engineer, without the need for high level expertise in the control domain.

Appendix A

Grid5000 basics

1. **Grid5000 login.**

   In order to login to Grid5000 you first need to generate a SSH key pair. Using Linux you can generate the SSH key using the following command:

   ssh-keygen -t rsa -P ""

   This command will generate two files into /home/userame/.ssh/ directory. id_rsa is the private key and id_rsa.pub is the public key.

   The content of the public key needs to be added to your grid5000 account.

   `https://api.grid5000.fr/sid/users/_admin/index.html`

   After you've added the public key you can connect to grid5000 from your computer using the following command:

   ssh grid5000username@access.grid5000.fr

2. **Reserving a cluster**

   You log in, with SSH, to the site where you want to run the experiments (Grenoble for example). From there you run a command such as this:

   oarsub -I -t allow_classic_ssh -l nodes=8,walltime=1:00:00 -p "cluster='genepi'"

   oarsub -I creates an interactive reservation, where you can run multiple jobs. The default reservation is 8 nodes. The command returns a numeric unique ID that identifies your submission. The -I option automatically connects you to the job's first node.

   Current submission can be viewed at

   `https://www.grid5000.fr/mediawiki/index.php/Status#Monika`.

   For example, to check the submissions state of the cluster at Grenoble go to:

   `https://intranet.grid5000.fr/oar/Grenoble/monika.cgi`

   or, by using Ganglia , to:

   `https://intranet.grid5000.fr/ganglia/?r=day&s=descending&c=` .

3. **oarstat** - Lists current job submissions.

4. **oarnodes** - Lists cluster properties.

5. **oarprint** - A tool for printing the current job resources.
   oarstat -j OAR_JOB_ID -p | oarprint core -P host,cpuset,memcore -F "%[%] (%)" -f - | sort

Appendix B

Using the MapReduce Benchmark Suite

The basic steps of launching an MRBS experiments are:

Step.1 Download MRBS from:
http://sardes.inrialpes.fr/research/mrbs/

Step.2 Copy the downloaded MRBS folder into to the main cluster node folder from where you desire to run the experiments.

Step.3 Go into your mrb/properties directory and configure the following files: mrbs.properties, mrbs.faultload, mrbs.workload.

Step.4 Execute the following command to run an MRBS experiment:
java -cp mrbs.jar:./hadoop/hadoop-0.20.2-core.jar:$(echo lib/.jar | tr ' ' ':') mrb.benchmark.server.MRBServer mrbs.output properties/mrbs.properties_ interactive properties/mrbs.faultload-nofaults properties/mrbs.workload*

Remember to copy your private key to the /home/username/.ssh directory of your local cluster so that MRBS won't ask for password while connecting to the nodes.

References

[1] Abdelzaher, T.F. et al. "Feedback performance control in software services." In: *Control Systems, IEEE* 23.3 (2003), pp. 74–90 (cit. on p. 24).

[2] Ahmad, Faraz et al. "Tarazu: optimizing MapReduce on heterogeneous clusters." In: *SIGARCH Comput. Archit. News* 40.1 (Mar. 2012), pp. 61–74 (cit. on p. 23).

[3] Ali-Eldin, Ahmed, Tordsson, Johan, and Elmroth, Erik. "An adaptive hybrid elasticity controller for cloud infrastructures." In: *NOMS*. IEEE, 2012, pp. 204–212 (cit. on p. 20).

[4] Ali-Eldin, Ahmed et al. "Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control." In: *Proceedings of the 3rd Workshop on Scientific Cloud Computing Date*. ScienceCloud '12. Delft, The Netherlands: ACM, 2012, pp. 31–40 (cit. on p. 20).

[5] Amazon. *AmazonEMR*. `http://aws.amazon.com/elasticmapreduce/` (cit. on pp. 1, 28).

[6] *Amazon Elastic Compute Cloud*. `http://aws.amazon.com/ec2/` (cit. on p. 20).

[7] Årzén, K. E. "A Simple Event-Based PID Controller." In: *Preprints of the 14th World Congress of IFAC*. 1999 (cit. on p. 63).

[8] Aström, K. J. "Event Based Control." English. In: *Analysis and Design of Nonlinear Control Systems*. Ed. by Astolfi, Alessandro and Marconi, Lorenzo. Springer Berlin Heidelberg, 2008, pp. 127–147 (cit. on p. 62).

[9] Aström, K. J. and Eykhoff, P. "System identification-A Survey." In: *Automatica* 7.2 (Mar. 1971), pp. 123–162 (cit. on pp. 14, 15, 51).

[10] Åström, K. J. and Wittenmark, B. *Adaptive control*. Courier Corporation, 2013 (cit. on pp. 52, 53).

[11] Battré, Dominic et al. "Nephele/PACTs: a programming model and execution framework for web-scale analytical processing." In: *Proceedings of the 1st ACM symposium on Cloud computing*. ACM. 2010, pp. 119–130 (cit. on pp. 2, 24).

[12] Behm, Alexander et al. "Asterix: towards a scalable, semistructured data platform for evolving-world models." In: *Distributed and Parallel Databases* 29.3 (2011), pp. 185–216 (cit. on pp. 2, 24).

[13] Berekmeri, Mihaly et al. In: *Proceedings of IFAC World Congress*. 2014 (cit. on p. 66).

[14] Cappello, F. et al. "Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed." In: *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*. Washington, DC, USA, 2005, pp. 99–106 (cit. on p. 74).

[15] Cardosa, Michael et al. "STEAMEngine: Driving MapReduce provisioning in the cloud." In: *18th International Conference on High Performance Computing (HiPC)*. Bengalore, India, 2011, pp. 1–10 (cit. on pp. 2, 22).

[16] Casalicchio, Emiliano and Silvestri, Luca. "Mechanisms for SLA provisioning in cloud-based service providers." In: *Computer Networks* 57.3 (2013), pp. 795–810 (cit. on p. 8).

[17] Cerf, Sophie. *Controle des Systemes Distribues - Application a MapReduce*. Tech. rep. Ecole Centrale de Lyon, 2015 (cit. on pp. 53, 80, 102).

[18] Chen, Yanpei, Alspaugh, Sara, and Katz, Randy H. *Design Insights for MapReduce from Diverse Production Workloads*. Tech. rep. UCB/EECS-2012-17. EECS Department, University of California, Berkeley, 2012 (cit. on p. 2).

[19] Compuware-ResearchInAction. *The hidden costs of managing applications in the cloud*. `http://www.hyperlinkki.mediaparkki.com/wp-content/uploads/2013/08/WP_CostofCloud.pdf` (cit. on p. 27).

[20] Dean, Jeffrey and Ghemawat, Sanjay. "MapReduce: simplified data processing on large clusters." In: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on pp. 1, 10).

[21] Dittrich, Jens et al. "Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing)." In: *Proc. VLDB Endow.* 3.1-2 (Sept. 2010), pp. 515–529 (cit. on p. 23).

[22] Durand, S. and Marchand, N. "Further Results on Event-Based PID Controller." In: *Proceedings of the European Control Conference*. 2009 (cit. on pp. 63, 64).

[23] Evolven. *Downtime, Outages and Failures - Understanding Their True Costs*. 2013 (cit. on pp. 2, 85).

[24] Ferguson, Andrew D. et al. "Jockey: Guaranteed Job Latency in Data Parallel Clusters." In: *Proceedings of the 7th ACM European Conference on Computer Systems*. EuroSys '12. Bern, Switzerland: ACM, 2012, pp. 99–112 (cit. on pp. 21, 22).

[25] Filieri, Antonio et al. "Software Engineering Meets Control Theory." In: *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. Firenze, Italy, May 2015 (cit. on pp. 2, 25, 26).

[26] Gandhi, Anshul et al. "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers." In: *ACM Trans. Comput. Syst.* 30.4 (Nov. 2012), 14:1–14:26 (cit. on p. 20).

[27] García, Andrés García, Espert, Ignacio Blanquer, and García, Vicente Hernández. "SLA-driven dynamic cloud resource management." In: *Future Generation Computer Systems* 31 (2014), pp. 1–11 (cit. on p. 8).

[28] Gong, Zhenhuan, Gu, Xiaohui, and Wilkes, John. "PRESS: PRedictive Elastic ReSource Scaling for cloud systems." In: *CNSM*. IEEE, 2010, pp. 9–16 (cit. on p. 20).

[29] Guillermo J, Silva. *PID Controllers for Time-Delay Systems*. Birkhauser Boston, 2005 (cit. on pp. 45, 58).

[30] Guitart, Jordi, Torres, Jordi, and Ayguade, Eduard. "A survey on performance management for internet applications." In: *Concurrency and Computation: Practice and Experience* 22.1 (2010), pp. 68–106 (cit. on p. 25).

[31] Hadoop. *Hadoop 1.1.2 Release Notes*. `http://hadoop.apache.org/docs/r1.1.2/releasenotes.html` (cit. on pp. 10, 74).

[32] Hellerstein, Joseph L et al. *Feedback control of computing systems*. John Wiley & Sons, Inc., New Jersey, 2004 (cit. on pp. 2, 24, 25).

[33] Herodotou, Herodotos. "Hadoop Performance Models." In: *CoRR* abs/1106.0940 (2011) (cit. on p. 20).

[34] Herodotou, Herodotos and Babu, Shivnath. "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs." In: *Proc. of the VLDB Endowment* 4.11 (2011), pp. 1111–1122 (cit. on p. 23).

[35] Jin, Hui et al. "ADAPT: Availability-Aware MapReduce Data Placement for Non-dedicated Distributed Computing." In: *ICDCS*. IEEE, 2012, pp. 516–525 (cit. on p. 23).

[36] Jin, Hui et al. "Performance under Failures of MapReduce Applications." In: *CCGRID*. 2011, pp. 608–609 (cit. on p. 21).

[37] Kavulya, Soila et al. "An Analysis of Traces from a Production MapReduce Cluster." In: *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGRID)*. Washington, DC, USA, 2010, pp. 94–103 (cit. on p. 21).

[38] Lim, Harold C., Babu, Shivnath, and Chase, Jeffrey S. "Automated Control for Elastic Storage." In: *Proceedings of the 7th International Conference on Autonomic Computing*. ICAC '10. Washington, DC, USA: ACM, 2010, pp. 1–10 (cit. on p. 25).

[39] Lin, Heshan, Ma, Xiaosong, and Feng, Wu-Chun. "Reliable MapReduce computing on opportunistic resources." In: *Cluster Computing* 15.2 (June 2012), pp. 145–161 (cit. on p. 2).

[40] Lin, Xuelian et al. "A practical performance model for hadoop mapreduce." In: *Cluster Computing Workshops, 2012 IEEE International Conference on*. IEEE. 2012, pp. 231–239 (cit. on p. 21).

[41] Ljung, Lennart. "Prediction Error Estimation Methods." In: *Circuits, systems, and signal processing* 21.1 (2002), pp. 11–21 (cit. on pp. 14, 51).

[42] Ljung, Lennart. *System Identification*. Wiley Encyclopedia of Electrical and Electronics Engineering, 1999 (cit. on p. 50).

[43] Luenberger, D.G. "Observers for multivariable systems." In: *IEEE Transactions on Automatic Control* 11 (1966), pp. 190 –197 (cit. on p. 71).

[44] Malrait, L., Marchand, N., and Bouchenak, S. "Modeling and Control of Server Systems: Application to Database Systems." In: *Proceedings of the European Control Conference (ECC)*. Budapest, Hungary, 2009, pp. 2960–2965 (cit. on p. 25).

[45] Marchand, N., Durand, S., and Guerrero-Castellanos, J. F. "A general formula for event-based stabilization of nonlinear systems." In: *IEEE Transactions on Automatic Control* 58.5 (2013), pp. 1332–1337 (cit. on pp. 61, 62).

[46] Matlab. *pidtool*. `http://fr.mathworks.com/help/control/ref/pidtool.html` (cit. on p. 58).

[47] Matsunaga, Andréa and Fortes, José A. B. "On the Use of Machine Learning to Predict the Time and Resources Consumed by Applications." In: *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. CCGRID '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 495–504 (cit. on p. 20).

[48] Microsoft. *HDInsight*. `http://azure.microsoft.com/en-us/services/hdinsight/` (cit. on pp. 1, 28).

[49] Nathuji, Ripal, Kansal, Aman, and Ghaffarkhah, Alireza. "Q-clouds: Managing Performance Interference Effects for QoS-aware Clouds." In: *Proceedings of the 5th European Conference on Computer Systems*. EuroSys '10. Paris, France: ACM, 2010, pp. 237–250 (cit. on p. 20).

[50] Nguyen, Hiep et al. "AGILE: Elastic Distributed Resource Scaling for Infrastructure-as-a-Service." In: *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 13)*. San Jose, CA: USENIX, 2013, pp. 69–82 (cit. on pp. 20, 59).

[51] Ogata, K. *Discrete Time Control Systems*. Ed. by Englewood Cliffs, NJ. Prentice-Hall, 1995 (cit. on p. 72).

[52] Padala, Pradeep et al. "Automated Control of Multiple Virtualized Resources." In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys '09. Nuremberg, Germany: ACM, 2009, pp. 13–26 (cit. on p. 25).

[53] Parekh, Sujay S. et al. In: *Integrated Network Management*. Ed. by Pavlou, George, Anerousis, Nikos, and Liotta, Antonio. IEEE, pp. 841–854 (cit. on p. 25).

[54] Patikirikorala, T. et al. "A systematic survey on the design of self-adaptive software systems using control engineering approaches." In: *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*. 2012, pp. 33–42 (cit. on pp. 2, 25).

[55] Poussot-Vassal, C., Tanelli, M., and Lovera, M. "Linear Parametrically Varying MPC for combined Quality of Service and energy management in Web service systems." In: *American Control Conference (ACC), 2010*. Baltimore, MD, 2010, pp. 3106–3111 (cit. on p. 25).

[56] Project. *AMADEOS*. `http://amadeos-project.eu/` (cit. on p. 102).

[57] Project. *HARNESS*. `http://www.harness-project.eu/` (cit. on p. 8).

[58] Project. *MYCLOUD*. `http://mycloud.inrialpes.fr/` (cit. on p. 8).

[59] Quiane-Ruiz, Jorge-Arnulfo et al. "RAFTing MapReduce: Fast recovery on the RAFT." In: *Proceedings of the 27th International Conference on Data Engineering*. Washington, DC, USA, 2011, pp. 589–600 (cit. on p. 23).

[60] Rao, Sriram et al. "Sailfish: A Framework for Large Scale Data Processing." In: *Proceedings of the Third ACM Symposium on Cloud Computing*. SoCC '12. San Jose, California: ACM, 2012, 4:1–4:14 (cit. on p. 23).

[61] Rizvandi, Nikzad Babaii, Taheri, Javid, and Zomaya, Albert Y. "Network Load Analysis and Provisioning of MapReduce Applications." In: *CoRR* abs/1206.2016 (2012) (cit. on p. 21).

[62] Roffel, B. and Betlem, B. *Advanced Practical Process Control*. Advances in soft computing. Springer, 2004 (cit. on p. 60).

[63] Rutten, E. et al. "Control of autonomic computing systems." Submitted to ACM Computing Surveys. 2013 (cit. on p. 25).

[64] Sangroya, Amit, Serrano, Damián, and Bouchenak, Sara. "Benchmarking Dependability of MapReduce Systems." In: *IEEE 31st Symposium on Reliable Distributed Systems (SRDS)*. Irvine, CA, 2012, pp. 21 –30 (cit. on pp. 2, 10, 24).

[65] Schroeder, Bianca, Wierman, Adam, and Harchol-Balter, Mor. "Open Versus Closed: A Cautionary Tale." In: *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*. NSDI'06. San Jose, CA: USENIX Association, 2006, pp. 18–18 (cit. on pp. 10, 37).

[66] Serrano, Damian et al. "{SLA} guarantees for cloud services." In: *Future Generation Computer Systems* 0 (2015), pp. – (cit. on p. 8).

[67] Seuret, A., Prieur, C., and Marchand, N. "Stability of nonlinear systems by means of event-triggered sampling algorithms." In: *Journal of Mathematical Control and Information* (2013) (cit. on pp. 61, 62).

[68] Sharma, Bikash et al. "MROrchestrator: A Fine-Grained Resource Orchestration Framework for MapReduce Clusters." In: *IEEE CLOUD*. Ed. by Chang, Rong. IEEE, 2012, pp. 1–8 (cit. on p. 22).

[69] Shen, Yushi. *Enabling the New Era of Cloud Computing: Data Security, Transfer, and Management*. 1st. Hershey, PA, USA: IGI Global, 2013 (cit. on p. 1).

[70] Söderström, T. and Stoica, P. *System identification*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1988 (cit. on p. 51).

[71] Stepan Shevtsov, Mihaly Berekmeri and Weyns, Danny. *The literature review supporting material available online*. `http://homepage.lnu.se/staff/daweaa/SLR/FSE/CBAS.htm` (cit. on p. 102).

[72] Thomas, Coleman, Branch, Mary Ann, and Grace, Andrew. *Coleman, Thomas, Mary Ann Branch, and Andrew Grace. Optimization Toolbox for Use with MATLAB: User's Guide, Version 2*. Math Works, Incorporated. 1999 (cit. on p. 68).

[73] Tian, Chao et al. "A Dynamic MapReduce Scheduler for Heterogeneous Workloads." In: *Proceedings of the 8th International Conference on Grid and Cooperative Computing (GCC)*. Washington, DC, USA, 2009, pp. 218–224 (cit. on p. 2).

[74] Tian, Fengguang and Chen, Keke. "Towards Optimal Resource Provisioning for Running MapReduce Programs in Public Clouds." In: *IEEE International Conference on Cloud Computing (CLOUD)*. Washington, DC, USA, 2011, pp. 155–162 (cit. on pp. 2, 21).

[75] Urgaonkar, Bhuvan, Shenoy, Prashant, and Roscoe, Timothy. "Resource Overbooking and Application Profiling in Shared Hosting Platforms." In: *SIGOPS Oper. Syst. Rev.* 36.SI (Dec. 2002), pp. 239–254 (cit. on p. 20).

[76] Verma, Abhishek, Cherkasova, Ludmila, and Campbell, Roy H. "ARIA: automatic resource inference and allocation for mapreduce environments." In: *Proceedings of the 8th ACM international conference on Autonomic computing*. ICAC '11. Karlsruhe, Germany: ACM, 2011, pp. 235–244 (cit. on pp. 2, 22).

[77] Verma, Abhishek, Cherkasova, Ludmila, and Campbell, RoyH. "Resource Provisioning Framework for MapReduce Jobs with Performance Goals." In: *Middleware 2011*. Vol. 7049. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 165–186 (cit. on pp. 2, 21).

[78] Vianna, Emanuel et al. "Analytical Performance Models for MapReduce Workloads." English. In: *International Journal of Parallel Programming* 41.4 (2013), pp. 495–525 (cit. on p. 21).

[79] Villegas, Norha M. et al. "A Framework for Evaluating Quality-driven Self-adaptive Software Systems." In: *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. SEAMS '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 80–89 (cit. on p. 25).

[80] Wang, Guanying et al. "Using realistic simulation for performance analysis of mapreduce setups." In: *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*. LSAP '09. Garching, Germany: ACM, 2009, pp. 19–26 (cit. on p. 24).

[81] Wang, Kewen, Lin, Xuelian, and Tang, Wenzhong. "Predator; An experience guided configuration optimizer for Hadoop MapReduce." In: *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*. Taipei, Taiwan, 2012, pp. 419–426 (cit. on p. 2).

[82] White, Tom. *Hadoop: the definitive guide*. O'Reilly Media, CA, 2012 (cit. on p. 10).

[83] Xu, Lijie. "MapReduce Framework Optimization via Performance Modeling." In: *IEEE 26 International Parallel & Distributed Processing Symposium (IPDPS)*. Shanghai, China, 2012, pp. 2506–2509 (cit. on p. 21).

[84] Yang, Hailong et al. "MapReduce Workload Modeling with Statistical Approach." English. In: *Journal of Grid Computing* 10 (2 2012), pp. 279–310 (cit. on pp. 21, 38).

[85] Yang, Xiao and Sun, Jianling. "An analytical performance model of MapReduce." In: *Cloud Computing and Intelligence Systems (CCIS), 2011 IEEE International Conference on*. 2011, pp. 306–310 (cit. on p. 21).

[86] Yfoulis, Christos A. and Gounaris, Anastasios. *Honoring SLAs on cloud computing services: a control perspective*. 2009 (cit. on p. 25).

[87] Zaharia, Matei et al. "Improving MapReduce performance in heterogeneous environments." In: *Proceedings of the 8th USENIX Conference on Operating systems design and implementation (OSDI)*. San Diego, California, 2008, pp. 29–42 (cit. on p. 2).

[88] Zaharia, Matei et al. "Spark: cluster computing with working sets." In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. 2010, pp. 10–10 (cit. on pp. 2, 24).