1-28-2015

Optimization and Regulation of Performance for Computing Systems

Jose Marcio Luna Castaneda

José Marcio Luna Castañeda

Candidate

Electrical and Computer Engineering Department

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Chaouki Abdallah , Chairperson

Gregory Heileman

Rafael Fierro

Jens Lorenz

Optimization and Regulation of Performance for Computing Systems

by

José Marcio Luna Castañeda

B.Sc., Electronics Engineering, Universidad Distrital Francisco José de Cáldas, 2004

M.Sc., Electrical Engineering, University of New Mexico, 2010

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

December, 2014

Dedication

A mi familia

A mi abuelita Blanca

A la memoria de Mamita Hely

Acknowledgments

I would like to express my sincere gratitude to my advisor, Dr. Chaouki T. Abdallah for his guidance, understanding, sincerity and support during all these years. I am really proud of being his student.

I would like to thank my dissertation committee for their support in the culmination of this work. Thanks to Dr. Gregory Heileman, Dr. Rafael Fierro, and Dr. Jens Lorenz.

To my wife, Roberta Arruda for her patience, support and company which were essential to accomplish my studies.

To Aura E. Castañeda Lobo, José Moisés Luna Rondón, Claudia Lorena Luna Castañeda, Juan Diego Luna Castañeda and David Ernesto Luna Luna, to whom I dedicate this work.

Thanks to Blanca Stella Lobo Turizo, my beloved grandmother.

Special thanks to my friend Joel Becktell for doing so much for me.

To my old friends Mauricio Soto, Sebastian Patrón, Luz Marcela Carolina Ayala, Héctor Cristyan Manta, and Javier Ulises González, my soul brothers.

To all my friends in Albuquerque, life was definitely better because of you.

Optimization and Regulation of Performance for Computing Systems

by

José Marcio Luna Castañeda

B.Sc., Electronics Engineering, Universidad Distrital Francisco José de Cáldas, 2004

M.Sc., Electrical Engineering, University of New Mexico, 2010

Ph.D., Engineering, University of New Mexico, 2014

Abstract

The current demands of computing applications, the advent of technological advances related to hardware and software, the contractual relationship between users and cloud service providers and current ecological demands, require the refinement of performance regulation on computing systems. Powerful mathematical tools such as *control systems theory*, *discrete event systems* (DES) and *randomized algorithms* (RAs) have offered improvements in efficiency and performance in computer scenarios where the traditional approach has been the application of well founded common sense and heuristics.

The comprehensive concept of computing systems is equally related to a microprocessor unit, a set of microprocessor units in a server, a set of servers interconnected in a data center or even a network of data centers forming a cloud of virtual resources. In this dissertation, we explore theoretical approaches in order to optimize

and regulate performance measures in different computing systems. In several cases, such as cloud services, this optimization would allow the fair negotiation of *service level agreements* (SLAs) between a user and a cloud service provider, that may be objectively measured for the benefit of both negotiators.

Although DES are known to be suitable for modeling computing systems, we still find that traditional control theory approaches, such as passivity analysis, may offer solutions that are worth being explored. Moreover, as the size of the problem increases, so does its complexity. RAs offer good alternatives to make decisions on the design of the solutions of such complex problems based on given values of confidence and accuracy.

In this dissertation, we propose the development of: a) a methodology to optimize performance on a many-core processor system, b) a methodology to optimize and regulate performance on a multitier server, c) some corrections to a previously proposed passivity analysis of a market-oriented cloud model, and d) a decentralized methodology to optimize cloud performance. In all the aforementioned systems, we are interested in developing optimization methods strongly supported on DES theory, specifically *Infinitesimal Perturbation Analysis* (IPA) and RAs based on sample complexity to guarantee that these computing systems will satisfy the required optimal performance on the average.

Contents

List of Figures **xii**

List of Tables **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 Thesis Statement 5

1.3 Contributions 6

1.4 Organization 7

2 Performance Optimization and Regulation for Many-core Processors **9**

2.1 Introduction 9

2.2 Previous Work: Throughput Regulation 11

2.2.1 IPA for Throughput Regulation 13

2.2.2 Regulation Algorithm 14

2.3 Regulation of Additional Performance Metrics through IPA 14

2.3.1 Estimate of Average System Time Derivative 15

2.3.2 Estimate of Average Waiting Time Derivative 15

2.4 Statistical Learning for Optimal Reference 16

2.5 Case Study: Energy Savings and Wait States 20

2.5.1 Performance Function 21

2.6 Simulation Results 22

2.7 Conclusions 27

3 Performance Optimization and Regulation for Multitier Servers 31

3.1 Introduction 31

3.2 A Multitier Model Based on Queue Networks 34

3.3 IPA for Performance Regulation 37

3.3.1 Unbiasedness of IPA Estimators 38

3.3.2 Sample Function for Throughput and its Derivative 42

3.4 Statistical Learning for Optimal Parameterization 49

3.4.1 Performance Function 51

3.5 Simulation Results 52

3.6 Conclusions 53

4 Cloud Computing Model with Time-Varying Workload 56

4.1 Introduction 56

4.2 Market-Oriented Cloud Model 58

4.3 More About Passivity Analysis 61

4.4 Effect of Equilibrium Points in Stability 64

4.5 Effect of Time-Varying $\mathbf{w}(k)$ 65

4.6 Simulation Results 67

4.7 Conclusions 68

5 Resource and Security Provisioning on the Cloud 71

5.1 Introduction 71

5.2 Probabilistic Performance Analysis 73

5.2.1 Worst-case Performance for Finite Families 73

5.3 Resource Optimization in the Cloud 77

5.3.1 Security Metrics based on Cryptography 78

5.3.2 RA for Optimization 80

5.3.3 Heuristics for Execution Time Reduction 83

5.4 Experimental Verification 87

5.4.1 Experimental Results 89

5.5 Conclusions 93

6 Concluding remarks, future work and recommendations 99

6.1 Performance Optimization and Regulation for Many-Core Processors 100

6.1.1 Concluding Remarks 100

6.1.2 Future Work and Recommendations 100

6.2 Performance Optimization and Regulation for Multitier Servers . . . 101

6.2.1 Concluding Remarks 101

6.2.2 Future Work and Recommendations 102

6.3 Market-oriented Cloud Model with Time-varying Workload 102

6.3.1 Concluding Remarks 102

6.3.2 Future Work and Recommendations 103

6.4 Optimal Performance and Security Provisioning in the Cloud 103

6.4.1 Concluding Remarks 103

6.4.2 Future Work and Recommendations 104

A Event Time Derivatives for Three-Tier Servers 106

B Recalculation of $\hat{u}(k)$ for Market-oriented Cloud 108

C Heuristic Execution Time Reduction for Security and Performance Optimization in the Cloud 110

References 112

List of Figures

1.1 Block diagram of interaction between the plant, the open-loop optimizer and the closed-loop regulator. 6

2.1 Block diagram of out-of-order execution core. 12

2.2 Open-loop optimizer and closed-loop regulator in the many-core processor problem. 17

2.3 Regulated average waiting time s_k for all four cores in the simulation. 26

2.4 Regulated throughput y_k for all four cores in the simulation. 27

2.5 Regulated average waiting time w_k for all four cores in the simulation. 28

2.6 Frequency signal ϕ_k for all four cores in the simulation. 29

2.7 Step-size term in the gradient descent algorithm, $K_k \frac{e_{n-1_k}}{y'_k(\phi)}$ for all four cores in the simulation. 29

2.8 Performance function J_k for all four cores in the simulation. 30

3.1 Multitier server in series. 33

3.2 Model of a multitier server. 35

3.3 Open-loop optimizer and closed-loop regulator in the multitier server problem. 37

3.4 State transition diagram of the two-tier server model. 44

3.5 Model of a one-tier server. 46

3.6 Model of a three-tier server. 49

3.7 Plot of regulated throughput for a three-tier server. 53

3.8 Plot of the controlled parameter μ_1 for a three-tier server. 54

3.9 Plot of the regulation error of the throughput for a three-tier server. 55

4.1 Block diagram of the market-oriented cloud presented in [1]. 58

4.2 Non-asymptotically stable example that satisfies the sufficient conditions for asymptotic stability given in [1]. 64

4.3 Simulation showing that the market-oriented cloud model in [1] is ISS. 68

4.4 Simulation showing bounded $d(k)$ with random and bounded $w(k)$. . 69

4.5 Simulation showing bounded $b(k)$ with random and bounded $w(k)$. . 70

4.6 Simulation showing bounded $s(k)$ with random and bounded $w(k)$. . 70

5.1 Example step 1. The available configurations 020, 107 and 210 (encircled) block the selection of the adjacent remaining entries (highlighted). 85

5.2 Example step 2. The previously selected configurations are discarded (crossed out) and the new available ones are 115 and 200. 86

5.3 Example step 3. The previously selected configurations are discarded and the repeated configuration 120 is the last available one. 88

5.4 CPU utilization for all three users in Experiment # 1. 92

5.5 Memory utilization for all three users in Experiment # 1. 93

5.6 Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 1. 94

5.7 CPU utilization for all three users in Experiment # 2. 95

5.8 Memory utilization for all three users in Experiment # 2. 96

5.9 Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 2. 96

5.10 CPU utilization for all three users in Experiment # 3. 97

5.11 Memory utilization for all three users in Experiment # 3. 97

5.12 Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 3. 98

List of Tables

5.1 Measure of security associated to ciphers and modes of operation . . 79

5.2 Example: Number of cycles of Algorithm 6 and Instance Configurations 87

5.3 Coefficients for Multi-objective Function in Experiments 90

Chapter 1

Introduction

1.1 Motivation

The widespread use of computer applications in areas such as business, research and entertainment have increased the demand for computational resources such as processing, storage capability and memory size. The development and posterior popularization of data networks has encouraged the development of efficient communication protocols and methodologies to, among other things, reduce latency and improve throughput in network applications. Thanks to recent advances in networking, very demanding processing tasks may be carried out by a group of computers working in parallel in *data centers*. Moreover, virtualization technology along with the current networking services available have made possible the interconnection of data centers to offer virtualized resources to remote users through the services of the *cloud*.

In a common scenario, a user runs an application from a remote location. This could be accomplished by using a remote *multiprocessor computer*, by accessing a remote *server* or by taking advantage of virtual resources available in the *cloud*,

among other possibilities. Notice that these three scenarios are part of a bottom-up structure with different levels of complexity where the cloud operates on a group of data centers, in turn, data centers contain groups of processors which usually contain multiple cores.

Power consumption in computing systems have become a major budget concern. In large data centers about 23-50% of the income should be invested on energy [2]. In fact, up to 40% of the technology budget of a company covers the cost of energy [3] since for every 1 W of power spent on the operation of servers, 0.5–1 W of additional power are required for the cooling equipment [4]. Furthermore, IT produces around 2% of global CO_2 emissions, an amount equivalent to the emissions of global air traffic. Therefore, the search for mechanisms to reduce power consumption have become a very relevant topic given its potential economical and environmental benefits. However, the reduction of power consumption may have negative effects on the processing performance of computing system.

Recently, cloud computing services have become the paradigm of large scale infrastructure where a third party provides computational services through shared virtual computing and storage resources to a client [5, 6]. The use of the third party infrastructure translates into cost reductions for the client who does not invest in infrastructure and maintenance. However, the fact that the interactions between clients and infrastructure are carried out through shared computer networks, has raised serious concerns about security, trust and privacy [7, 8]. *Service Level Agreements* (SLAs) are the required documents that define the relationship between a service provider and a client or recipient [9, 10, 11]. These documents provide the description of the contractual commitments of both parties, focusing mainly on the desired performance of the service. SLAs are supported over performance metrics known as *Server Level Objectives* (SLOs), such as, desired response time, availability and reliability of the system. Until very recently, cloud security was not considered

in the SLAs because of the difficulties to quantify security levels. The cloud community has pointed out that by specifying security in SLAs, it facilitates the modeling and assessment of the security on the provided services [12]. The authors in [13, 14] proposed quantitative mechanisms to asses cloud security levels based on *Reference Evaluation Methodology* (REM) and *Quantitative Policy Trees*. These security metrics are our starting point to incorporate security provisioning in a unified approach to optimize performance and provide security in the cloud.

Due to the complexities of the aforementioned computing systems, heuristic approaches are commonly proposed to solve the complex problems of power consumption reduction, performance optimization and regulation, and security provision. In the last few years, control theory has had a productive but yet limited relationship with computing theory and systems [15, 16]. Control theory is being used in problems such as managing power consumption for microprocessors [17, 18], data centers [2, 19, 20, 21] and managing resources in cloud computing applications [10, 22] among others. The traditional approach assumes an available model that encompasses the main features of the phenomena to be controlled. Assuming an operative model, the controller designer proceeds to develop mathematical tools to obtain "well-behaved" systems, *i.e.,* systems that allow a convenient control of the outputs based on the excitation at the inputs of the system. Depending on the particular goals of the controller, the output of the problem could be regulated to a reference value, the states of the system could track a trajectory, reject disturbances or reach specific values in finite or infinite time, among the variety of options offered by control theory.

Traditional control theory assumes deterministic models of the plants that are defined through *differential equations* for *continuous-time systems* and *difference equations* for *discrete-time systems* [23]. This approach has been used in several computing problems [6, 24, 25]. However, obtaining the dynamic equations of computing systems is not always possible. Thus, a common approach is to use *model*

identification [26]. It consists of assuming a parameterized mathematical function of the model, and by using *adaptive filtering* techniques the parameters are estimated based on the measured inputs and outputs of the system. Since computing systems have proven to be essentially time-varying [6] the parameters should be calculated continuously, and this implies an overhead on the performance of the actual controller, which may be significant even if the structure of the model is assumed to be time-invariant and linear.

Computing systems are an example of complex technological systems that are governed by operational rules controlled and designed by humans [23, 27]. As a consequence, rather than being time driven, as in the case of systems governed by differential or difference equations, they may be modeled as driven by asynchronous and discrete *events* such as pushing a bottom, sending a message packet or a random system failure. These systems are known as *discrete event systems* (DES) [28]. For all the aforementioned problems related to performance regulation and optimization, DES theory offers powerful mathematical tools that allow efficient theoretical analysis to guarantee stability and regulation of the different performance measures on the average. *Sensitivity analysis* allows the evaluation of the effect of a parameter in the behavior of a DES automaton. *infinitesimal perturbation analysis* (IPA) is a powerful tool for sensitivity analysis that allows the estimation of the derivatives of performance functions in DES automata from a sample path taken at the output of the automata. By being able to calculate these estimates, we may use a gradient descend optimization approach to minimize a cost function that is directly proportional to a regulation error as in [17, 29, 30]. The simplicity of the implementation of IPA algorithms is suitable for real-time applications, however, its further development has been hindered by the limited spectrum of problems where unbiased estimates may be guaranteed. However, further experimental evidence has proven that even with biased but bounded estimates, IPA may be enough to solve more complex problems.

Randomized algorithms (RAs) have been previously proposed to solve robust control design problems and have proven useful and implementable in a wide variety of NP-hard problems. RAs are based on *sample complexity* and *tail inequalities* and are the basis of *statistical learning theory* [31]. RAs take advantage of powerful results associated with *Monte Carlo simulations* and the *uniform law of large numbers*. Necessary conditions have been proposed to design efficient RAs to estimate a cost function whose closed form is not available. These algorithms are not guaranteed to work all the time, but most of the time [32], because the probability that the algorithm fails cannot be made identically zero. In complex systems such as many-core processors, multitier servers or cloud computing services, RAs offer several possibilities to optimize performance given the intrinsic randomness of DES and the sometimes non-convex nature of the performance metrics involved.

We are interested in regulating and optimizing performance in computing systems at different complexity levels. We focus on three different levels namely a) microprocessor level, b) multitier server level and c) cloud computing level. Notice that each level can be understood as a cluster containing a set of elements of the previous one. This dissertation aims to propose methods for performance regulation and optimization using mainly feedback techniques based on sensitivity analysis and open-loop solutions based on RAs as illustrated in Fig. 1.1.

1.2 Thesis Statement

This PhD dissertation proposes the implementation of mathematically rigorous performance regulation and optimization techniques for computing systems at different levels of complexity, namely, microprocessor level, multitier level and cloud computing level. The main goal is to develop a formal mathematical approach to optimize, and under certain assumptions, regulate hardware performance to a desired value on

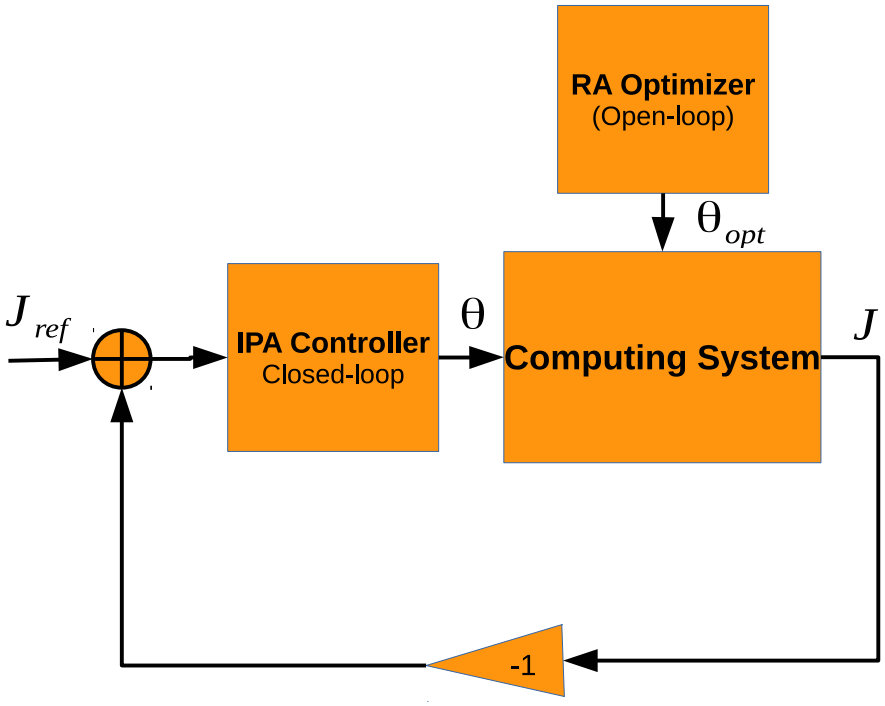

Figure 1.1: Block diagram of interaction between the plant, the open-loop optimizer and the closed-loop regulator.

the average. RAs, DES theory and IPA are the main mathematical tools to guarantee that the aforementioned computing systems will satisfy the required optimal performance on the average.

1.3 Contributions

A list of the main contributions of this dissertation includes:

- Development of a mathematical background based mainly on DES modeling,

IPA and RAs to regulate multiple performance metrics in many-core processors and multitier servers.

- The formal introduction of a multi-objective optimization approach to automate the calculation of optimal parameter values for regulation of performance metrics at the microprocessor level and the multitier level.
- A theoretically justified approach to optimize virtual resources in the cloud while provisioning security, with special emphasis on its application in SLA negotiations.
- The validation of the theoretical results through simulations in the microprocessor and multitier levels.
- The validation of the theoretical results through experimentation at the cloud level.

1.4 Organization

This doctoral research specializes in the development and adaptation of mathematical solutions to the problem of optimization and regulation of performance of computing systems. Four problems have been identified that determine the organization of this dissertation as follows: Chapter 2 presents a theoretical approach to regulate many-core processor systems using statistical learning and IPA. This includes the analysis of a case study based on a real processors model supported by simulation results. Chapter 3 proposes an approach to apply statistical learning along with IPA to regulate performance on a previously validated queue model of a multitier server. Simulation results of the optimization and regulation of multiple performance measures on a three-tier server are presented. Chapter 4 presents a detailed stability analysis of a previously proposed market-oriented cloud model. An additional

sufficient condition for asymptotic stability and a proof that the system is *input-to-state-stable* (ISS) are presented. These results have been previously published in [33]. Chapter 5 presents an RA based on sample complexity for finite families to optimize virtual resources in the cloud. This approach is validated through an implementation using Amazon Web Services, Amazon Elastic Compute Cloud (AWS EC2). This work has been submitted to [34]. Finally, chapter 6 presents the conclusions and future work.

Chapter 2

Performance Optimization and Regulation for Many-core Processors

2.1 Introduction

Given the power limitations of current microprocessor architectures, the exponential performance growth obeying Moore's Law [35] and Dennard's scaling [36] has stalled in recent times. The transition from single core processors to multiple core processors offered a short-term solution to this issue, by increasing the throughput of the processors and by using several cores working in parallel but at lower frequencies. Subsequently, the transition from *in-order* to *out-of-order* cores produced an additional improvement in performance and in the implementation of affordable chip power envelopes. Out-of-order cores implement instruction-level parallelism and speculative execution, thus relaxing the order of execution while increasing the throughput. However, some limitations we must deal with are the increase of the

power consumption and area. The strong pipeline implemented in these cores, affects the predictability of the system, in detriment of its energy efficiency. To reduce the chip power consumption, many-core processors were introduced. Power consumption remains however the main limitation to increase processor speed while keeping affordable architectures [37, 38].

Dynamic voltage frequency scaling (DVFS) has been the most successful and accurate mean to minimize and regulate chip power consumption [39, 40]. Reducing frequency and voltage in a processor translates into cubic power reductions [38]. Nowadays, most of the commercial processors incorporate per-core DVFS capabilities, which allow the implementation of decentralized and scalable performance controllers in many-core processors. Throughput regulation proved to be beneficial in real-time applications such as, video streaming and processing, as well as improving the predictability and energy efficiency of the system [41, 42, 43, 44].

In the recent work in [17], the authors propose DVFS in order to modify the frequency of a multiprocessor system thus regulating throughput. The approach assumes that the throughput is regulated to a previously chosen reference value. In this dissertation, we propose a mathematical approach that allows the automatic calculation of the optimal reference value of the throughput. Moreover, we propose the regulation of the *average system time* and *average waiting time* per core, in addition to regulating throughput. In this way, the many-core processors may be defined as a multiple-input-multiple-output (MIMO) system that allows the regulation of linear or nonlinear combinations of several performance measures.

Along the same lines of [17], we implement a DES approach [28] using feedback control techniques through IPA. Finally, we propose the use of *sample complexity* based on *statistical learning* theory [31] to calculate the optimal reference value of performance. This chapter is organized as follows: In Section 2.2, we present the previous work on throughput regulation using IPA presented in [17]. In Section 2.3,

we introduce the mathematical equations to regulate the average system time and the average waiting time at each core using IPA. In Section 2.4, we introduce the concept of statistical learning and present an RA to calculate an optimal reference value for performance regulation. In Section 2.5, we present a case study where a four-core processor is interfaced with a slower peripheral. In Section 2.6, we present some simulation results based on our case study. In Section 2.7 we present our conclusions.

2.2 Previous Work: Throughput Regulation

Almoosa *et al.,* present a throughput regulation method based on IPA in [17]. A descriptive scheme of the out-of-order execution core is shown in Fig. 2.1. The following equations model the dynamics of the k-th core,

$$a_{i_k} = l_k(i)\theta_k, \tag{2.1}$$

$$\alpha_{i_k} = \max\{a_{i_k}, \delta_{\kappa(i_k)}\} + \theta_k, \tag{2.2}$$

$$\delta_{i_k} = \begin{cases} \alpha_{i_k} + n_{i_k}\theta_k, & \text{sync. instructions or cache memory fetch} \\ \alpha_{i_k} + T_{\text{mem}_k}, & \text{other memory fetches} \end{cases}, \tag{2.3}$$

$$d_{i_k} = \max\{\delta_{i_k} + \theta_k, d_{i-1_k}\} + \theta_k, \tag{2.4}$$

where the index i denotes the index of the i-th instruction arriving to the k-th core, a_{i_k} denotes the enqueue time, $l(i_k)$ represents the clock-cycle count of the enqueue time a_{i_k}. θ_k is the clock period of the processor, α_{i_k} denotes the issue time of the instruction to be processed after arrival, δ_{i_k} represents the complete time of the instruction after processing and d_{i_k} the dequeue time of the instruction.

Since the i-th instruction may need data from a previous instruction that has not yet been completed, let us denote $\kappa(i_k)$ the index of the previous instruction that

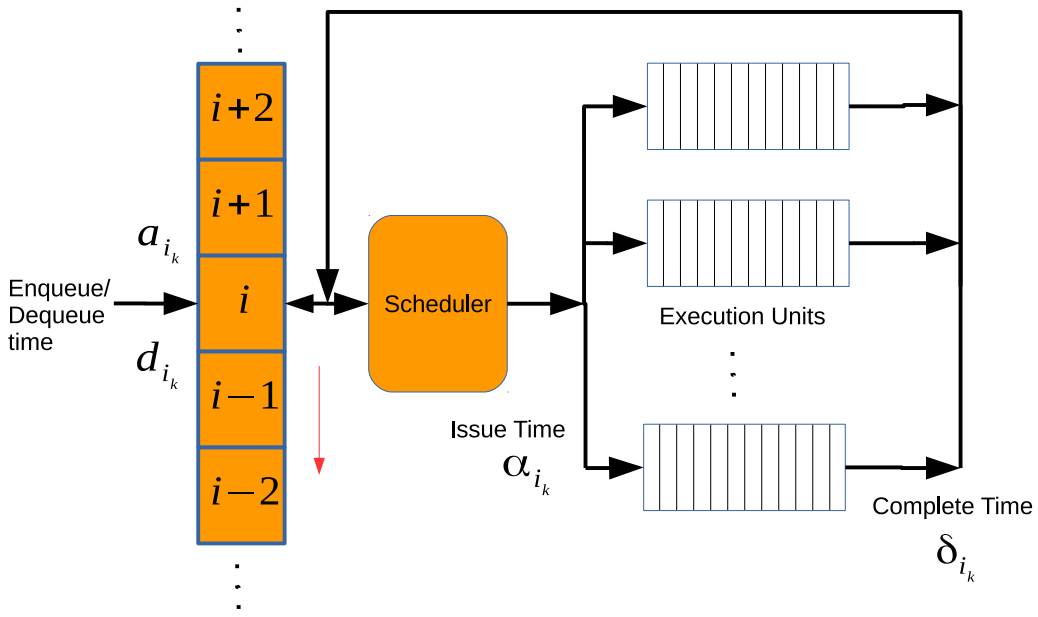

Figure 2.1: Block diagram of out-of-order execution core.

produces the results to be used by the i-th instruction. The i-th instruction may be executed as a synchronous instruction or a cache memory fetch. In the first case, the time it takes to be processed is modeled as a multiple of an integer number less than or equal to 10 denoted by $n(i)$. If the instruction is executed as a memory fetch (or other kind of asynchronous fetch), then the time to process it is modeled as $T\text{mem}_k$ which is equal to period θ_k of the processor multiplied by a factor which depends on the benchmark problem, usually equal to several hundreds. The frequency of the processor ϕ_k is given by $\phi_k = 1/\theta_k$. Finally, the throughput y_k is estimated by,

$$y_k = \frac{M_k}{d_{M_k}}, \tag{2.5}$$

for a given integer M_k which indexes the last processed instruction in the k-th core.

2.2.1 IPA for Throughput Regulation

Since the system requires high speed to compute a controller that regulates the throughput, the ability of IPA to carry out *sensitivity analysis* of random signals with few calculations offers an alternative to implementing gradient optimizers in real time. However, IPA has the disadvantage of providing statistically-biased gradients even for simple systems [45]. There is experimental evidence that suggests that biased IPA estimates may be used in not so trivial applications as long as the bias is bounded [27, 46]. Given the stochastic nature of the performance metrics that can be extracted from the many-core processors, their derivative may be estimated using IPA. The following proposition provides the equations to estimate the derivative of the dequeue time with respect to the frequency of a processor core.

Proposition 1 *Given the DES model (2.1)–(2.4), the following equations apply for the k-th core with* $i = 1, \ldots, M_k$*,*

$$\begin{aligned} \alpha'_{i_k}(\theta_k) &= \begin{cases} \alpha'_{\kappa(i)_k}(\theta_k) + v_k(\kappa(i)) + 1, & \text{if } I_i \text{ stalls upon arrival} \\ l_k(i) + 1, & \text{if } I_i \text{ does not stall upon arrival.} \end{cases} \\ d'_{i_k}(\theta_k) &= \alpha'_{m(i)_k}(\theta) + v_k(m(i)) + i - m_k(i) + 2, \end{aligned} \tag{2.6}$$

where,

$$\begin{aligned} v_k(i) &= \begin{cases} 0, & \text{if } I_i \text{ is a memory fetch that is not from cache} \\ n_{i_k}, & \text{otherwise,} \end{cases} \\ m_k(i) &= \max\{m_k(i) \leq i : I_m \text{ did not stall following its execution}\}. \end{aligned} \tag{2.7}$$

Proof The proof is provided in [17]. □

With the estimate of the derivative $d'_{i_k}(\theta)$, the authors estimate the derivative of the throughput with respect to the processor frequency as,

$$y'_k(\phi_k) = \frac{1}{M_k}\left(\frac{y_k}{\phi_k}\right)^2 d'_{M_k}(\theta_k). \tag{2.8}$$

2.2.2 Regulation Algorithm

The expression in (2.8) allows for the calculation of an integral control gain, which may be interpreted as an application of a gradient descent method to minimize the performance function of the k-th core given by the square of the regulation error:

$$(e_k)^2 = (y_{ref_k} - y_k)^2,$$

where y_{ref_k} is the required throughput. Finally, the update of the frequency of the k-th core is given by the following equation,

$$\phi_{n_k} = \phi_{n-1_k} + K_k \frac{e_{k_{n-1}}}{y'_k(\phi_k)} \tag{2.9}$$

where $K_k \in \mathbb{R}$ is a positive scalar that determines the step-size of the gradient descend method along with $e_{k_{n-1}}$ and $y'_k(\phi_k)$. The index variable $n = 1, 2 \ldots$, represents the discrete time instants.

2.3 Regulation of Additional Performance Metrics through IPA

Given the recursive equations for $d'_{i_k}(\theta)$ in Proposition 1, we proceed to calculate additional performance measures as functions of the departure time d_{i_k}. In this section, we present our first contribution by introducing the IPA equations to calculate the average system time and the average waiting time of each instruction in a processor core.

2.3.1 Estimate of Average System Time Derivative

The average system time s_k is defined as the average time that an instruction spends in the k-th core, *i.e.,* the average difference between the dequeue and enqueue time and it may be estimated by the equation,

$$s_k = \frac{U_{M_k}}{M_k} \tag{2.10}$$

where

$$U_{M_k} = \sum_{i=1}^{M_k} d_{i_k} - a_{i_k},$$

for some positive integer M_k. Therefore, the derivative with respect to ϕ_k gives,

$$s'_k(\phi_k) = -\frac{U'_{M_k}(\theta_k)}{M_k \phi_k^2},$$

then,

$$s'_k(\phi_k) = -\frac{\sum_{i=1}^{M_k} d'_{i_k}(\theta) - l_k(i)}{M_k \phi_k^2},$$

with $d'_{i_k}(\theta_k)$ given by (2.6).

2.3.2 Estimate of Average Waiting Time Derivative

The average waiting time w_k is the elapsed time between the arrival instant of an instruction and the time it starts being served, and it is given by,

$$w_k = \frac{U_{\gamma_k}}{M_k}, \tag{2.11}$$

with $\gamma_k = \{i \in \mathbb{Z} : d_{i-1_k} - a_{i_k} > 0\}$ and

$$U_{\gamma_k} = \sum_{i \in \gamma_k} d_{i-1_k} - a_{i_k}.$$

Therefore,

$$w'_k(\theta_k) = -\frac{M_k U'_{\gamma_k}(\theta_k)}{M_k^2 \phi_k^2} = -\frac{U'_{\gamma_k}(\theta_k)}{M_k \phi_k^2},$$

then,

$$w'_k(\theta_k) = -\frac{\sum_{i \in \gamma_k} d'_{i-1_k} - l_k(i)}{M_k \phi_k^2}.$$

2.4 Statistical Learning for Optimal Reference

The sensitivity analysis guarantees the regulation of performance measures using a closed-loop controller, and it is assumed that the reference value is provided *a priori*. In this section, we contribute an approach to automate the generation of appropriate reference values for regulation using *sample complexity* analysis based on statistical learning theory. This approach consists of designing a cost function to be minimized in order to get an optimal reference value. In this particular problem, the use of statistical learning aims at estimating an optimal frequency that leads to optimal performance.

Since the statistical learning section of the controller works in open loop, the system is not guaranteed to regulate the performance measures to the calculated values if the statistics change due to variations in the benchmark problem. Therefore, our controller consists of two stages, a) a statistical learning stage where the reference performance measures are calculated through an optimization process and b) an IPA regulation stage where the system guarantees that the performance measures stay in a neighborhood of the desired reference values. These stages are illustrated in Fig. 2.2.

A very useful inequality for the purposes of this chapter is Hoeffding's [31]. In this section, we present some results on sample complexity that are derived from this inequality.

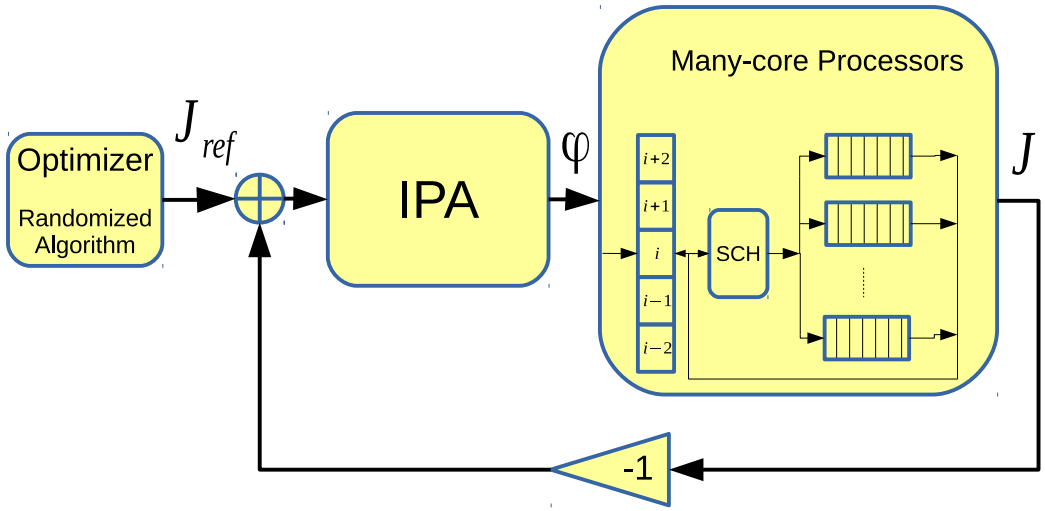

Figure 2.2: Open-loop optimizer and closed-loop regulator in the many-core processor problem.

Theorem 1 (Two-sided Hoeffding's inequality) *Consider a set of N independent random variables $\mathbf{x}_1, \ldots, \mathbf{x}_N$ such that $\mathbf{x}_i \in [a, b] \subset \mathbb{R}$ and define the new random variable $\boldsymbol{\varsigma}_N = \sum_{i=1}^{N} \mathbf{x}_i$. Then, for any $\epsilon > 0$,*

$$P_R\left\{|\boldsymbol{\varsigma}_N - \mathbb{E}(\boldsymbol{\varsigma}_N)| \geq \epsilon\right\} \leq 2e^{-\frac{2\epsilon^2}{N(b-a)^2}}, \tag{2.12}$$

where P_R denotes probability of the event in curly braces, and $\mathbb{E}(\boldsymbol{\varsigma}_N)$ denotes the expectation of the random variable ς_N.

Proof The proof is provided in [31]. □

In what follows, $\boldsymbol{\lambda} \in \boldsymbol{\Lambda} \subseteq \mathbb{R}^{n_\lambda}$ is a random variable with *probability distribution function* $f_{\boldsymbol{\lambda}}(\lambda)$ and $\boldsymbol{\lambda}^{(1,\ldots,N)}$ is a multi-sample of $\boldsymbol{\lambda}$. Let us consider the performance function $J : \boldsymbol{\Lambda} \to [0, 1]$ and calculate its empirical mean,

$$\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda})) = \frac{1}{N}\sum_{i=1}^{N} J(\boldsymbol{\lambda}^{(i)}),$$

From (2.12) we get that,

$$P_R\left\{\left|\mathbb{E}\left(J(\boldsymbol{\lambda})\right)-\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda}))\right|\geq\epsilon\right\}\leq 2e^{-2N\epsilon^2}. \tag{2.13}$$

Now, let us modify the performance function by defining a parameter vector $\boldsymbol{\psi}\in\boldsymbol{\Psi}\subseteq\mathbb{R}^{n_\psi}$ such that $J:\boldsymbol{\Lambda}\times\boldsymbol{\Psi}\rightarrow[0,1]$. Assuming that the set $\boldsymbol{\Psi}$ is finite with cardinality M, then we get a *finite family* [47] of functions.

$$\begin{aligned}\mathcal{J}_M &= \left\{J(\boldsymbol{\lambda},\boldsymbol{\psi}^{(1)}),\ldots,J(\boldsymbol{\lambda},\boldsymbol{\psi}^{(M)})\right\}\\ &= \left\{J(\boldsymbol{\psi}^{(1)}),\ldots,J(\boldsymbol{\psi}^{(M)})\right\},\end{aligned} \tag{2.14}$$

where $\boldsymbol{\psi}^{(1,\ldots,M)}$ is a multi-sample of $\boldsymbol{\psi}$.

The tail inequality in (2.13) applies for a single performance function J. However, the bound of the probability of deviation between the empirical and the actual mean of all the performance functions in the finite family (2.14) are calculated by applying the tail inequality (2.13) over and over M times, in order to obtain,

$$P_R\left\{\sup_{J\in\mathcal{J}_M}\left|\mathbb{E}\left(J(\boldsymbol{\lambda})\right)-\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda}))\right|\geq\epsilon\right\}\leq 2Me^{-2N\epsilon^2}. \tag{2.15}$$

Notice that as N goes to infinity in (2.15), the probability of deviation tends to zero asymptotically. However, we are interested in providing statements based on finite sample bounds which is the best we can do in real implementations. An RA aims at estimating the probability of fulfillment of a given performance specification. This estimate should be within a previously defined *accuracy* $\epsilon\in(0,1)$ from the current value with "high" confidence $1-\delta$, $\delta\in(0,1)$.

Given the uncertainties that $\boldsymbol{\lambda}$ incorporates into the performance function, a closed deterministic expression of $J(\boldsymbol{\lambda},\boldsymbol{\psi})$ is difficult to obtain. Hence, the best we can do is to try to calculate $\mathbb{E}\left(J(\boldsymbol{\lambda},\psi)\right)$. In general, the exact calculation of the expected value is computationally demanding since it usually involves the solution

of multiple integrals with non-convex domains of integration. Therefore, we proceed to calculate the empirical version,

$$\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda},\psi)) = \frac{1}{N}\sum_{i=1}^{N} J(\boldsymbol{\lambda}^{(i)},\psi).$$

Given a desired accuracy ϵ_1 and confidence $1-\delta_1$ we require that the estimate $\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda})$ satisfies,

$$P_R\left\{\sup_{J\in\mathcal{J}_M}\left|\mathbb{E}\left(J(\boldsymbol{\lambda})\right) - \hat{\mathbb{E}}_N(J(\boldsymbol{\lambda}))\right| \geq \epsilon_1\right\} \leq \delta_1.$$

To fulfill this requirement, (2.15) provides the sufficient condition $2Me^{-2N\epsilon_1^2} \leq \delta_1$ which implies that,

$$N \geq \frac{\ln\frac{2M}{\delta_1}}{2\epsilon_1^2}.$$

Therefore, given the accuracy ϵ_1, the confidence $1-\delta_1$ and the cardinality M of the finite parameter set $\boldsymbol{\Psi}$, we are able to determine the minimum number of random samples N needed to estimate the expectation of the performance function $\mathbb{E}\left(J(\boldsymbol{\lambda},\psi)\right)$. Notice that the RA may provide an erroneous estimate with probability at most δ_1. The following Theorem provides the basis for the performance optimization.

Theorem 2 *Given the empirical probable parameter vector,*

$$\hat{\boldsymbol{\psi}}_{M_1M_2} = \arg\min_{i=1,\ldots,M_2} \hat{\mathbb{E}}_{M_1}(J(\boldsymbol{\lambda},\boldsymbol{\psi}^{(i)})),$$

and the performance function $J:\boldsymbol{\Lambda}\times\boldsymbol{\Psi}\rightarrow[0,1]$ *with* $\boldsymbol{\lambda}\in\boldsymbol{\Lambda}\subset\mathbb{R}^n$, *and* $\boldsymbol{\psi}\in\boldsymbol{\Psi}\subset\mathbb{R}^m$, *for some given* $\epsilon_1,\epsilon_2,\delta\in(0,1)$, *let,*

$$M_2 \geq \frac{\ln\frac{2}{\delta}}{\ln\frac{1}{1-\epsilon_2}}, \tag{2.16}$$

and

$$M_1 \geq \frac{\ln \frac{4M_2}{\delta}}{2\epsilon_1^2}, \tag{2.17}$$

Then, with confidence $1-\delta$*, it holds that*

$$P_R\left\{\mathbb{E}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi})\right) < \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi}_{M_1M_2})\right) - \epsilon_1\right\} \leq \epsilon_2.$$

Proof The proof is given in [31]. □

Since all the assumptions of Theorem 2 are satisfied by the DES model of the many-core processor proposed in [17], we propose to use Algorithm 1 [27] to carry out the performance optimization. Now, we proceed to validate our approach by carrying out a simulation of a 4-core processor.

Algorithm 1 Performance Optimization

Define: $J : \boldsymbol{\Lambda} \times \boldsymbol{\Psi} \rightarrow [0,1]$

Define: $\epsilon_1, \epsilon_2, \delta \in (0,1)$

1: $M_2 \leftarrow \frac{\ln \frac{2}{\delta}}{\ln \frac{1}{1-\epsilon_2}}$ ▷ According to Theorem 2

2: $M_1 \leftarrow \frac{\ln \frac{4M_2}{\delta}}{2\epsilon_1^2}$

3: $\boldsymbol{\psi} \leftarrow \text{randSamples}(M_1)$ ▷ Draw M_1 samples of $\boldsymbol{\psi}$

4: $\boldsymbol{\lambda} \leftarrow \text{randSamples}(M_2)$ ▷ Draw M_2 samples of $\boldsymbol{\lambda}$

5: **return** $\hat{\boldsymbol{\psi}}_{M_1M_2} \leftarrow \arg\min_{i=1,\ldots,M_2} \frac{1}{M_1}\sum_{k=0}^{M_1} J(\boldsymbol{\lambda}^{(k)}, \boldsymbol{\psi}^{(i)})$

2.5 Case Study: Energy Savings and Wait States

In this case study, we assume that a microprocessor is exchanging data with a peripheral that runs at a slower clock frequency *e.g.,* an external memory. Although microprocessors have evolved to run at very high speeds, the speed of memories has

not grown at the same rate. The most common practice in these cases is to add *wait states* to the bus cycles. Wait states extend the processors read or write cycles by a number of clock cycles [48]. However, they are nothing but a waste of performance and some modern technical approaches such as *branch prediction*, *instruction prefetch* and *simultaneous multithreading* are aimed at reducing them, hiding them, or even eliminating them. Using our approach, we try to estimate a processor frequency that lowers the throughput to values in the operation range of the peripheral. Thus, we not only reduce the wait states of the processor, but by downscaling the frequency, we save processor power, while keeping low average system and waiting times for the instructions. Notice from (2.5), (2.10) and (2.11) that the throughput is directly proportional to the frequency, while the average system and waiting times are inversely proportional to the frequency.

2.5.1 Performance Function

Let us assume that we have a set of $\tilde{N}$ cores in a many-core processor. For the k-th core, we consider the performance metrics y_k, s_k and w_k given by (2.8), (2.10) and (2.11) respectively for optimization, with $k = 1, \ldots, \tilde{N}$.

Let us define the parameter set,

$$\mathbf{\Psi} = \left\{ \boldsymbol{\psi} \in \Phi^{\tilde{N}} : \boldsymbol{\psi} = (\phi_1, \ldots, \phi_{\tilde{N}}) \right\},$$

which consists of the vector containing the frequencies of all cores.

It is reasonable to assume that under normal operation of the k-th core in the processor, we can estimate finite bounds for all the performance measures, such that,

$y_k \in [y_{k_{\min}}, y_{k\max}]$, $s_k \in [s_{k_{\min}}, s_{k_{\max}}]$ and $w_k \in [w_{k_{\min}}, w_{k_{\max}}]$. Now, let us define,

$$\begin{aligned} \bar{y}_k &= \frac{y_k - y_{k_{\min}}}{y_{k_{\max}} - y_{k_{\min}}}, \\ \bar{s}_k &= \frac{s_k - s_{k_{\min}}}{s_{k_{\max}} - s_{k_{\min}}}, \\ \bar{w}_k &= \frac{w_k - w_{k_{\min}}}{w_{k_{\max}} - w_{k_{\min}}}, \end{aligned} \tag{2.18}$$

so that the range of all the normalized performance measures is $[0, 1]$.

Now, let us define the random vector,

$$\boldsymbol{\lambda}_k = (\bar{y}_k, \bar{s}_k, \bar{w}_k)^T \in \boldsymbol{\Lambda}, \tag{2.19}$$

and let us define the performance vector function,

$$\mathbf{J} = (J_1, \ldots, J_N)^T, \tag{2.20}$$

where $J_k(\boldsymbol{\lambda}, \boldsymbol{\psi}) = J_k$ for the k-th user is given by,

$$J_k = \alpha_k \bar{y}_k + \beta_k \bar{s}_k + \gamma_k \bar{w}_k, \tag{2.21}$$

where $\alpha_k, \beta_k, \gamma_k \in \mathbb{R}^+$ are chosen so that $J_k : \Phi \to [0, 1]$, with $\phi_k \in \Phi$.

2.6 Simulation Results

In this section, we present some simulation results using the DES model given by (2.1)–(2.4) along with the statistical learning approach explained in Section 2.4. These simulations were carried out using Matlab®. We base our simulated processor cores on the AMD Opteron processor which works in the frequency range $0.8 - 2.7$ GHz and a voltage range of $1.0 - 1.35$ V [49]. Let us assume that it exchanges communications with a real-time multi-media peripheral in the lower frequency range $0.1 - 2.5$ Ghz, which implies the presence of wait states in the communications.

We proceed to normalize all the performance measures as described in (2.18). Based on (2.20) and (2.21) we propose to minimize the following performance vector function,

$$\mathbf{J} = \begin{pmatrix} J_1 \\ J_2 \\ J_3 \\ J_4 \end{pmatrix} = \begin{pmatrix} \frac{1}{3}\bar{s}_1 + \frac{1}{3}\bar{y}_1 + \frac{1}{3}\bar{w}_1 \\ \frac{3}{5}\bar{s}_2 + \frac{1}{5}\bar{y}_2 + \frac{1}{5}\bar{w}_2 \\ \frac{5}{14}\bar{s}_3 + \frac{2}{7}\bar{y}_3 + \frac{5}{14}\bar{w}_3 \\ \frac{1}{3}\bar{s}_4 + \frac{2}{3}\bar{y}_4 + \frac{1}{3}\bar{w}_4 \end{pmatrix} \tag{2.22}$$

For this particular example we choose $\delta = \epsilon_1 = \epsilon_2 = 0.02$. Based on (2.16) and (2.17), in order to obtain a minimum for the multi-objective function (2.20) and (2.22) with accuracy 0.02 and confidence 0.98 we need to test at least $M_1 = 228$ samples of the frequency vector $\boldsymbol{\psi} = (\phi_1, \phi_2, \phi_3, \phi_4)^T$ and at least $M_2 = 13,410$ samples of the performance measurement vector (2.19) for each frequency vector sample.

One of the strengths of our approach is that every core may have independent priorities. This is reflected in the weights of the multi-objective function (2.22). Based on the definition of J_1, all three performance measurements have the same priority in the optimization process. In J_2, the minimization of the average system time $\bar{s}_2$ has more weight than the remaining performance metrics. In J_3 the minimization of $\bar{s}_3$ and $\bar{w}_3$ have higher priority than the minimization of $\bar{y}_3$. Lastly, in J_4 the minimization of the throughput $\bar{y}_4$ is more relevant than the minimization of the remaining performance measures. The optimal frequency $\boldsymbol{\phi}^*$ and the optimal performance function vector $\mathbf{J}^*$ obtained from Algorithm 1 are,

$$\begin{aligned} \boldsymbol{\phi}^* &= \left(1.5142 \times 10^9, \;\; 2.0097 \times 10^9, \;\; 2.0040 \times 10^9, \;\; 8.2734 \times 10^8\right)^T, \\ \mathbf{J}^* &= \left(0.1009, \;\; 0.09548, \;\; 0.1403, \;\; 0.1044\right)^T, \end{aligned} \tag{2.23}$$

and the related optimal performance vectors give,

$$\begin{aligned}
\mathbf{s}^* &= \left(2.2905 \times 10^{-7},\ \ 1.8237 \times 10^{-7},\ \ 1.8920 \times 10^{-7},\ \ 4.6386 \times 10^{-7}\right)^T, \\
\mathbf{y}^* &= \left(9.1114 \times 10^{8},\ \ 1.1570 \times 10^{9},\ \ 1.1103 \times 10^{9},\ \ 4.5532 \times 10^{8}\right)^T, \\
\mathbf{w}^* &= \left(2.2795 \times 10^{-7},\ \ 1.8151 \times 10^{-7},\ \ 1.8830 \times 10^{9},\ \ 4.6167 \times 10^{-7}\right)^T.
\end{aligned} \tag{2.24}$$

The optimal values of $\boldsymbol{\phi}^*$ and $\mathbf{J}^*$ are the references for the desired performance of the four cores. Nevertheless, statistical variations in the benchmark problems running in the processor may yield changes in the statistics of the cores. This implies that the optimization process is not enough to guarantee the desired performance. As another strength of our approach, and given the problem-specific priorities, we are able to regulate each one of the performance metrics of interest, namely, $\bar{s}_k, \bar{y}_k$ and $\bar{w}_k$ through DVFS. In fact, we are able to regulate linear or non-linear combinations of the performance function. For this example, let us assume that the reference values for regulation are organized in the normalized reference vector $\mathbf{r}$ as follows,

$$\mathbf{r} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{pmatrix} = \begin{pmatrix} \bar{s}_1^* \\ \frac{1}{2}\left(\bar{s}_2^* + \bar{y}_2^*\right) \\ \frac{1}{2}\left(\bar{s}_3^* + \bar{y}_3^*\right) \\ \bar{y}_4^* \end{pmatrix} \tag{2.25}$$

where $\bar{\mathbf{s}}^* = (\bar{s}_1^*, \ldots, \bar{s}_4^*)$, $\bar{\mathbf{y}}^* = (\bar{y}_1^*, \ldots, \bar{y}_4^*)$ and $\bar{\mathbf{w}}^* = (\bar{w}_1^*, \ldots, \bar{w}_4^*)$ are the normalized versions of the vectors (2.24) with $\bar{s}_k^*, \bar{y}_k^*, \bar{w}_k^* \in [0, 1]$ for $k = 1, \ldots, 4$. This normalization is necessary given the remarkable difference of orders among the optimal throughput $\bar{y}_k^*$ and the remaining variables.

In this example, we are assuming that on average, 20% of the arriving instructions depend on the results of another instruction, and that about 50% of the instructions are memory fetches in each core. This is of course, an extreme case but one that will

be useful for illustration purposes. Since we are simulating statistical variations in the processor, then by using a uniform random number generator we vary the average of dependent instructions between $10-30\%$, and the average memory fetches between $30-70\%$ of the total number of executed instructions. With a value of $K_k = 0.005$ in (2.9), we proceed to regulate the performance variables in all four cores. The results are presented in Fig. 2.3–2.5.

Fig. 2.3, presents the regulated average system time s_k for all four cores in the processor. Notice from (2.25), that the only case where the average system time is not explicitly regulated is in r_4 which corresponds to Core 4 (black line with pentagram markers) in the plot. In spite of the indirect regulation to its reference value due to its correlation with the other performance variables, the average system time of Core 4 does not exhibit the same transient behavior as in the plots for Cores 1, 2 and 3 in Fig. 2.3. Furthermore, from (2.22), the highest priority to minimize s_k is assigned to Core 2, which goes in accordance with the blue line with squared markers in Fig. 2.3 that illustrates the minimal average system time among all four cores.

From (2.22), the highest priority for regulating throughput y_k is assigned to Core 4, which corresponds to the black line with pentagram markers in Fig. 2.4. In fact, this line exhibits the minimum throughput among all four cores.

The average waiting time w_k is illustrated in Fig. 2.5. This variable is highly correlated to the average system time s_k based on (2.10) and (2.11). Although both variables do not have the same exact values, the plots in Fig. 2.3 and 2.5 illustrate their similarity. This raises the question of whether these two variables should be regulated together or separately. The answer to this question is out of the scope of this dissertation. However, regarding this dependency among variables, notice that in (2.25) we only regulate the average system s_k and the throughput y_k, taking advantage of the correlation between s_k and w_k.

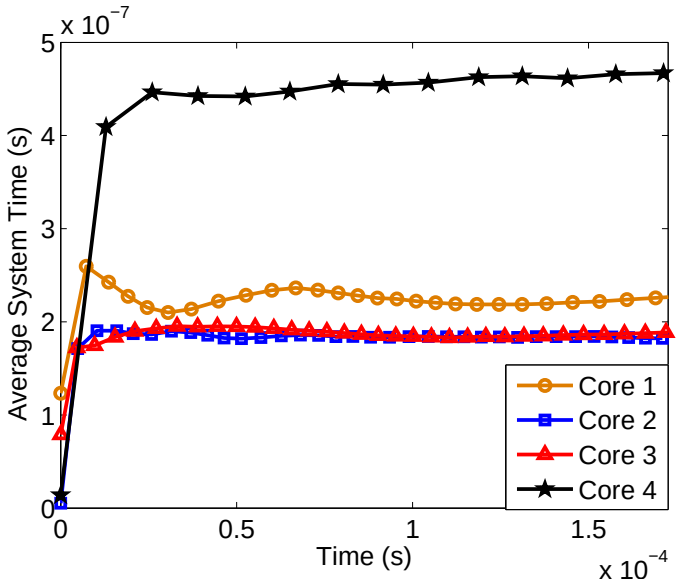

Figure 2.3: Regulated average waiting time s_k for all four cores in the simulation.

Since the main tool for implementing this controller is DVFS, we plot the variations of frequency (control signal) that keep this system regulated in Fig. 2.6. The variations in amplitude and period of the signal illustrate the adaptability of the controller as the system approaches the reference value.

The step-size term given by $\phi_n - \phi_{n-1}$ in (2.9) is illustrated in Fig. 2.7. The adaptability of the IPA controller is evident through the decrease in step-size term as the system approaches the reference value in all four processors. Finally, the regulated performance functions J_k are shown in Fig. 2.8. The values correspond to the optimal ones presented in (2.23).

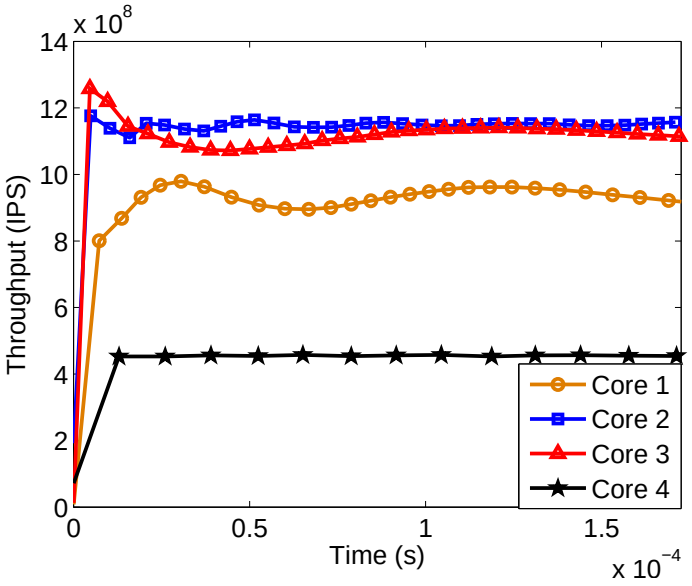

Figure 2.4: Regulated throughput y_k for all four cores in the simulation.

2.7 Conclusions

We presented a methodology to regulate multiple performance functions in a DES model of a many-core processor. The performance functions covered in this chapter are the throughput, the average system time and the average waiting time. A statistical learning approach is proposed to calculate the optimal frequency that satisfies the optimal performance of a multi-objective function. After the optimal performance is calculated, we carry out a regulation process through an integral control while estimating the derivatives online using IPA. This regulator keeps the performance of the closed-loop system in the vicinity of its optimal value.

We validated our approach through simulations using a simplified model of the microprocessor. We observed that if the regulated variables are highly correlated, such as in the case of average waiting time and the average system time, the difference between regulating one or both does not make a substantial difference as reflected in

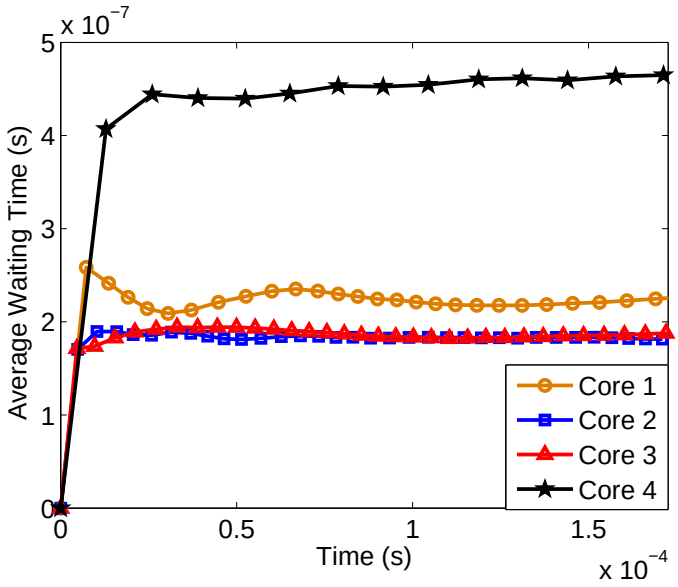

Figure 2.5: Regulated average waiting time w_k for all four cores in the simulation.

Fig. 2.3 and 2.5. However, this approach proves to be valuable for the decentralized regulation of a group of cores since a decentralized regulator is highly scalable. Its importance becomes more apparent as the number of cores embedded in a processor keeps increasing.

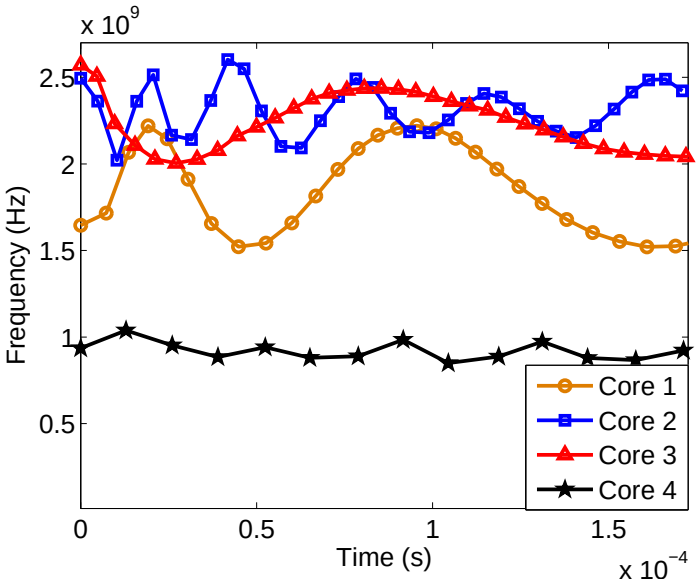

Figure 2.6: Frequency signal ϕ_k for all four cores in the simulation.

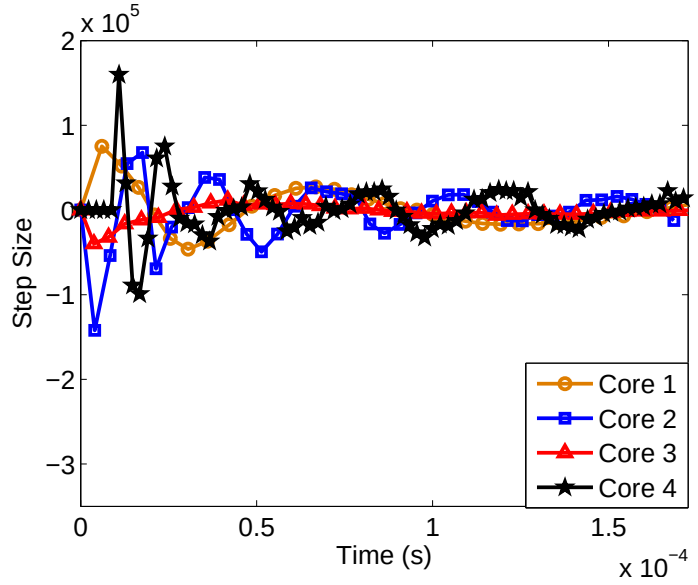

Figure 2.7: Step-size term in the gradient descent algorithm, $K_k \frac{e_{n-1_k}}{y'_k(\phi)}$ for all four cores in the simulation.

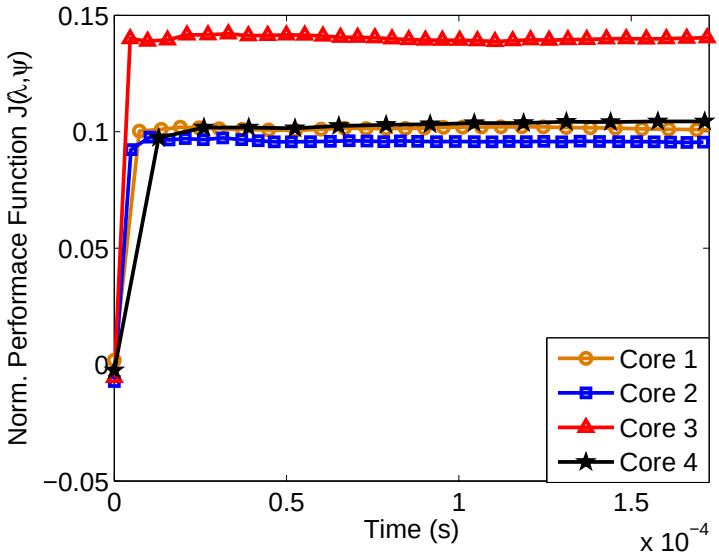

Figure 2.8: Performance function J_k for all four cores in the simulation.

Chapter 3

Performance Optimization and Regulation for Multitier Servers

3.1 Introduction

Common internet services such as, e-mail, online retails, news and e-commerce are based on client-server architectures where multiple clients access an online server concurrently. The dynamics of these services are difficult to model because of the randomness induced by the event-based interactions between clients and servers, as well as the amount of clients accessing the system and sharing web resources. Server *unavailability* and *thrashing* are undesired behaviors directly related to variations in heavy workloads usually present in these systems. *Admission control* is a common practice to improve the availability of internet services [50]. It consists of limiting the number of clients of the server by defining a *multi-programming level* (MPL) parameter. The MPL is a quantity that defines the maximum number of requests that may be processed by the server. Whenever a server reaches its concurrency limit, the subsequent requests are dropped generating an error message and activat-

ing a timeout mechanism that allows the reissue of the request for a number of times before the request is either processed or finally abandoned. This approach is mainly supported by heuristics, trial and error and ad-hoc tunning instead of formal theoretical concepts. This raises concerns about its lack of optimality since it is proven to have a strong effect on the *quality of service* (QoS), server performance and server availability [50, 51].

Internet applications employ multitier architectures distributed on a cluster of servers. Each tier provides a specific functionality which carries out a part of the overall request. Every tier uses the service provided by its successor and provides a service to its predecessor in order to process the overall request following the layout illustrated in Fig. 3.1. Multitier servers may have as many tiers as it is required. However, a typical server consists of three tiers namely, a *front-end tier* which usually carries out the HTTP tasks, a *middle tier* that implements core applications, (*e.g.,* Java enterprise server), and a *back-end tier* which implements a database server. In this case the MPL parameter is usually defined for each tier, and the contention control is carried out between tiers, therefore, every tier is able to drop requests from its predecessor once its MPL has been reached.

Mathematical modeling is a powerful tool to optimize, regulate and predict performance of systems with dynamic behavior. Linear differential and difference equations to describe multitier server behavior were proposed in [52]. However, the dynamics of multitier servers are intrinsically nonlinear [53], which makes the linear models limited for an accurate analysis of the system. Furthermore, computing systems such as the multitier server, are event-based rather than time-based, which makes the utilization of time-dependent differential equations not entirely suitable. DES models may provide an accurate description of the behavior of multitier servers [54, 55]. For some real-time applications, such as video streaming and remote monitoring, being able to improve the predictability of the system's behavior is important to get a reli-

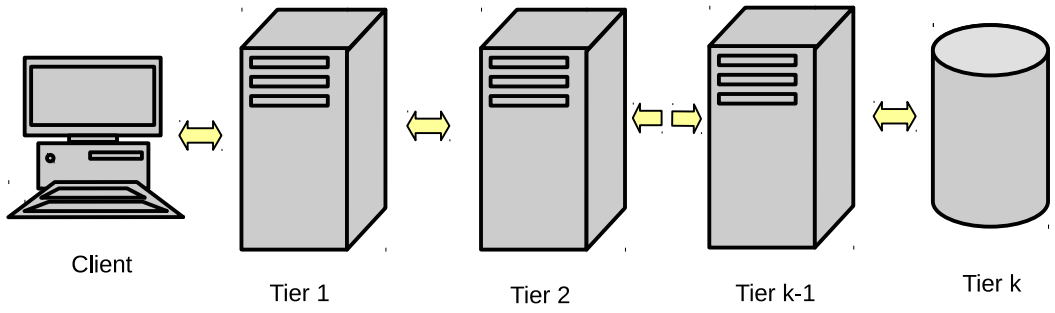

Figure 3.1: Multitier server in series.

able estimate of performance measures of the system and to guarantee the provision of the service at a consistent frequency.

Our regulation approach provides not only a way to reduce power while guaranteeing average performance but it may potentially improve the predictability of multitier servers. Although the main drawback of DES approaches is the complexity of the model calibration process and of their mathematical analysis [56], IPA estimators have proven rather easy to implement. However, the price to pay for this "easiness" of implementation is the difficulty of carrying out rigorous mathematical analyses, making it possible only for simple cases. Guaranteeing the unbiasedness of IPA estimates in fairly complex systems is still an open problem. These limitations have hindered IPA's further development and raised questions about its applicability [17]. Experimental evidence and current work in the area, suggested that unbiasedness may not be a necessary condition for the successful application of optimization and control algorithms in practical applications. In fact, such evidence indicates that

low-complexity estimators with bounded bias rather than unbiased estimators may be enough to guarantee the applicability of IPA in complex event-based automata.

Urgaonkar *et. al.,* present and validate an analytical model that captures the dynamics of multitier servers in [52]. Based on the aforementioned model, we propose to use mathematical tools such as sensitivity analysis and statistical learning theory to optimize and regulate performance in these servers. To the best of the author's knowledge, no existing solution has been proposed towards this goal.

This chapter is organized as follows: Section 3.2, explains in detail the multitier model based on queueing network proposed in [52]. Section 3.3, describes the sensitivity analysis carried out over the queue network and how this result may be used to regulate throughput in the server. We introduce some additional modeling assumptions to guarantee unbiased IPA estimates. We present algorithms to regulate throughput in servers with one and three tiers. In Section 3.4, we incorporate an open-loop stage to optimize the configuration of the multitier servers prior to the regulation process by using a statistical learning approach. In Section 3.5, we present a case study simulation to minimize mean service rates, and indirectly, to minimize power consumption while keeping the throughput of the multitier server regulated. In Section 3.6, we present our conclusions.

3.2 A Multitier Model Based on Queue Networks

The following model was proposed in [52]. Let us assume an application with R tiers modeled by a *closed queueing network* interconnected as shown in Fig. 3.2. Each queue Q_i with $i = 1, \ldots, R$ represents a tier. The system has a constant population of N requests moving through the queues, such that the network is able process a maximum number of N concurrent requests, and once a request is processed, a new request enters the system in order to replace the processed one. When a

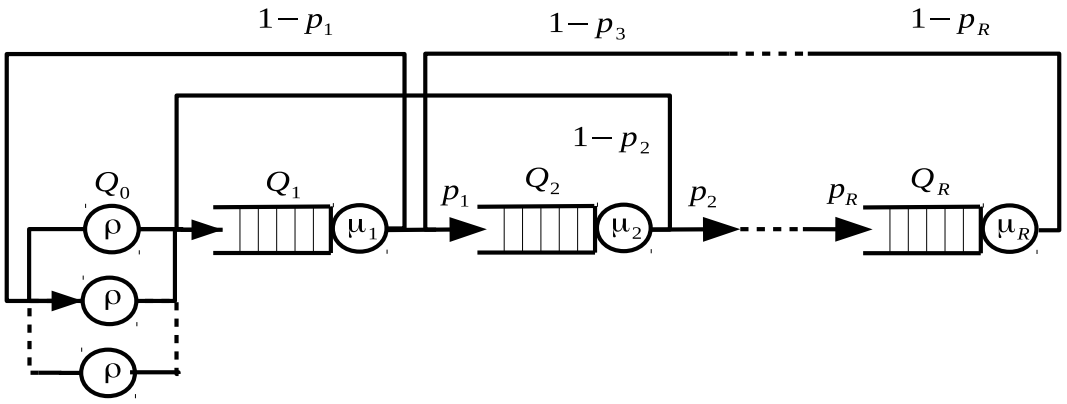

Figure 3.2: Model of a multitier server.

request arrives to tier Q_i, it activates one or more requests at tier Q_{i+1}. In real internet applications, a request may trigger parallel requests in some of the tiers, (*e.g.,* a keyword in an online store that searches several catalogs at the same time [52]). In this approach, parallel requests are modeled as sequential requests that visit the tiers multiple times. Therefore, there are transitions from tier Q_i to Q_{i-1} that allow every request to make multiple visits to every tier when processing the overall request. After a request has been processed by tier Q_i, it either proceeds to Q_{i+1} with probability p_i or returns to Q_{i-1} with probability $1 - p_i$. Notice that the last tier Q_R returns all its requests to queue Q_{R-1} and the first tier Q_1 completes its request every time there is a return to the preceding stage Q_0.

The first stage of the model, labeled Q_0, consists of an *infinite server queueing system* which incorporates the session-based nature of the internet workload. Every time an internet session is open, several requests are generated. This is modeled

by assuming sequential requests at Q_1 that start traveling back and forth between the tiers as required by the system until the overall request is processed and then it returns to Q_0. The processed request spends a so-called *think time* at Q_0 and after that, the following request of the same session enters the queueing network.

The think time is denoted by the random variable Y with $\mathbb{E}\{Y\} = \frac{1}{\rho}$. The random variable representing the service time of the i-th tier is denoted Z_i with $\mathbb{E}\{Z_i\} = \frac{1}{\mu_i}$, therefore, ρ represents the *mean think rate* and μ_i the *mean service rate* of the i-th tier. The mean service rate μ_i provides the average number of requests that are served in the i-th tier per unit of time, and is therefore an average measure of the operation frequency of every tier. The fact that it is modeled as a random variable conveys a) the idea that requests may be processed randomly in several cycles by the tiers, and b) the randomness induced in the tiers given the multiple open sessions being processed at every tier.

In the model proposed in [52], the authors assumed *product form closed queueing networks* to be able to calculate response times through the mean-value analysis (MVA) algorithm [57]. A broad class of queue networks are known to have product form solutions which is convenient for modeling a wide variety of multitier servers. Our goal is to tune the mean service rate in the front-end tier, namely, μ_1 so that, we are able to regulate the throughput to a reference value defined a priory by implementing a regulator based on IPA. In order to reduce power consumption, we propose an open-loop optimizer which reduces the values of the parameters μ_i with $i = 1, \ldots, R$, which are directly proportional to the frequency of the server processors, thus, by DVFS the power consumed by the server is reduced. In Fig. 3.3, we present a block diagram illustrating the open-loop optimizer and the closed-loop regulator.

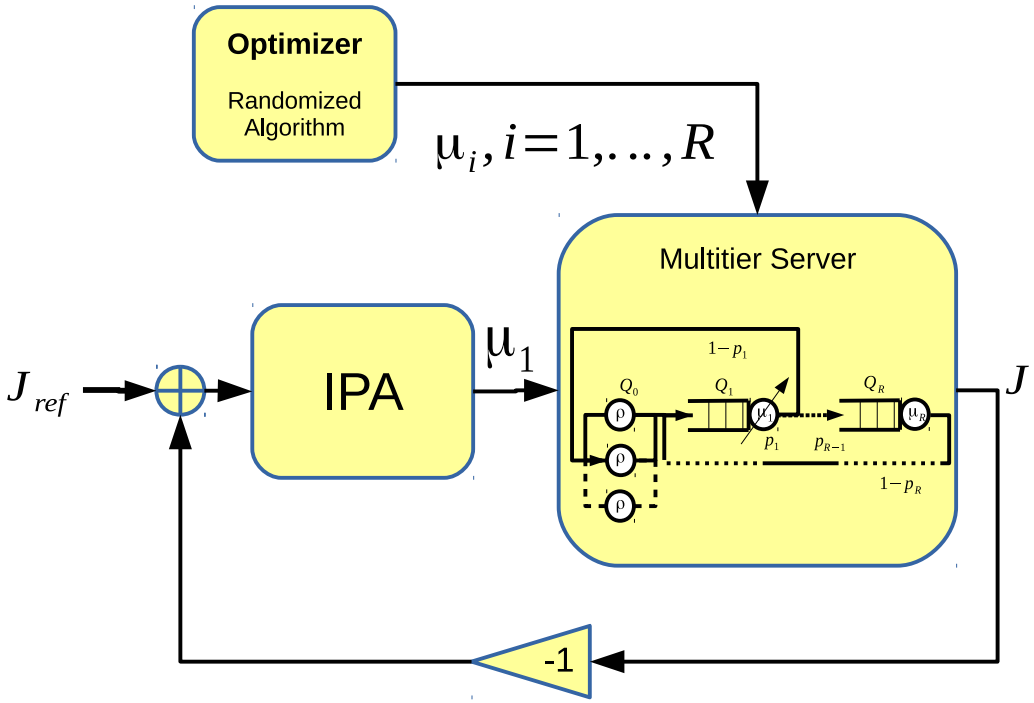

Figure 3.3: Open-loop optimizer and closed-loop regulator in the multitier server problem.

3.3 IPA for Performance Regulation

The product form closed queueing network assumption specified in [52] is key to guarantee that the MVA algorithm is applied to estimate performance measures of this queueing network. Since this is a closed queueing network formed by a set of R interconnected queues, the stationary state probability has the form

$$p(x_1, \ldots, x_R) = \frac{1}{C(N)} \prod_{i=1}^{R} f_i(x_i),$$

where $f_i(x_i)$ is a function of the state x_i of the i-th queue and $C(N)$ is a normalizing constant dependent on the request population size N. One of the advantages of this

particular model is its generality, since product form networks are not necessarily Markovian in nature. In fact, product form networks such as the BCMP network [27] may have several customer classes as well as a variety of possible queuing disciplines such as *first-come-first-served* (FCFS), *processor sharing* (PS), *last-come-first-served* (LCFS), and different service time distributions, among other complexities. The model in [52] was not defined considering sensitivity analysis applications. Therefore, we proceed to determine the sufficient conditions to guarantee the validity of IPA for the aforementioned model.

3.3.1 Unbiasedness of IPA Estimators

In this problem, we need to estimate the expectation of the derivative of a performance function $J : \mathbf{\Lambda} \times \mathbf{\Psi} \to \mathbb{R}$, such that, $J(\psi) = \mathbb{E}\{L(\lambda, \psi)\}$, where $L(\lambda, \psi)$ is the sample function of interest, *i.e.,*

$$\frac{dJ(\psi)}{d\psi} = \mathbb{E}\left\{\frac{dL(\lambda, \psi)}{d\psi}\right\}, \tag{3.1}$$

where $\lambda \in \mathbf{\Lambda} \subseteq \mathbb{R}^{n_\lambda}$ is a random variable with *probability distribution function* $f_{\boldsymbol{\lambda}}(\lambda)$ and $\lambda^{(1,\ldots,N)}$ is a multi-sample of λ. $\psi \in \mathbf{\Psi} \subseteq \mathbb{R}^{n_\psi}$ is the parameter vector. This means that the sample derivative is an unbiased estimate of $\frac{dJ}{d\psi}$. First of all, let us analyze the continuity of the sample function $L(\lambda, \psi)$. A necessary condition for continuity is the *commuting condition* (CC).

Commuting Condition

Let us use the convention $p(y|x, \alpha)$ to denote the probability that the queueing network goes to state y after the feasible event α is observed while being at state x. Now, we proceed to present the main result of commuting condition based on [27].

Theorem 3 *Let us consider a queue network whose state space is denoted by $\mathcal{X}$ and whose event set is denoted by Ξ. Let us define $\Gamma(x) \subset \Xi$ as the set of feasible events when the current state is $x \in \mathcal{X}$. Now, let $x, y, z_1 \in \mathcal{X}$ and $\alpha, \beta \in \Gamma(x)$ such that,*

$$p(z_1|x, \alpha) \cdot p(y|z_1, \beta) > 0.$$

Then, for some $z_2 \in \mathcal{X}$, we get that,

$$p(z_2|x, \beta) = p(y|z_1, \beta),$$

and

$$p(y|z_2, \alpha) = p(z_1|x, \alpha).$$

Moreover, for any $x, z_1, z_2 \in \mathcal{X}$ such that $p(z_1|x, \alpha) = p(z_2|x, \alpha) > 0$, we get $z_1 = z_2$.

Proof The proof is provided in [58]. □

The CC is not necessarily fulfilled by all product form queueing networks. However, there is a subset of such networks that has been proven to satisfy the CC. These networks are known as *Jackson-like networks*.

Jackson-like Networks [27]

Definition 1 *Jackson-like networks are open or closed queueing networks such that, every queue has one server with infinite queueing capacity and each queue implements a FCFS queue discipline. Furthermore, these networks process a single class of customers and the routing of costumers between queues is probabilistic.*

Event Time Derivatives

In order to derive expressions for event time derivatives we proceed to make the following assumptions based on [27].

Assumption 1 *Let us denote the lifetime of the k-th occurrence of the event $\alpha \in \Xi$ by $V_{\alpha,k}(\psi)$, where Ξ is the set of feasible events. Let us assume that for all event α, $V_{\alpha,k}(\psi)$ is almost surely continuously differentiable in ψ, with $k = 1, 2, \ldots$.*

Assumption 2 *For all event $\alpha \in \Xi$, with cumulative distribution function (cdf) $F_\alpha(x, \psi)$, with parameter $\psi \in \mathbf{\Psi}$ and $x \in \mathcal{X}$. Let us assume that $F_\alpha(x, \psi)$ is continuous in ψ and $F_\alpha(0, \psi) = 0$.*

Given the event α with associated event lifetime distribution $F_\alpha(x, \psi)$ with parameter ψ, we can define the lifetimes as functions of ψ, *i.e.,* $V_{a,k}(\psi)$. If Assumptions 1 and 2 are fulfilled, the derivative $\frac{dV_{\alpha,k}}{d\psi}$ may be calculated as,

$$\frac{dV_{\alpha,k}}{d\psi} = -\frac{\partial F_\alpha(x,\psi)/\partial\psi}{\partial F_\alpha(x,\psi)/\partial x}\bigg|_{x=V_{\alpha,k}}. \tag{3.2}$$

Although closed expressions can be calculated for some probability distributions using (3.2), some expressions can be easily determined by taking advantage of the definitions of scale and location parameters [27].

Definition 2 *Given two random variables X_1 and X_2 with cdfs $F_{X_1}(x_1, \psi_1)$ and $F_{X_2}(x_2, \psi_2)$ respectively, where $\psi_1, \psi_2 \in \mathbf{\Psi}$, we say that ψ_1 is a* scale parameter *if the cdf of $\psi_1 X_1$ is independent of ψ_1. Furthermore, we say that ψ_2 is a* location parameter *if the cdf of $X_2 - \psi_2$ is independent of ψ_2.*

As an example, the normal distribution has a location and a scale parameter, namely, the mean and the standard deviation respectively. In another example, the exponential distribution has a scale parameter which is the inverse of its mean. The

Cauchy distribution has a location parameter which corresponds to its mean. Based on Definition 2 and (3.2), it is easy to prove that if a random variable X_1 has scale parameter ψ_1 then,

$$\frac{dX_1}{d\psi_1} = \psi_1 X_1. \tag{3.3}$$

Moreover, if a random variable X_2 has a location parameter ψ_2 then,

$$\frac{dX_2}{d\psi_2} = 1.$$

A general-purpose algorithm for evaluating event time derivatives while a sample path is observed in a General-Semi-Markov-Process (GSMP) is presented in [27]. We reproduce it in Algorithm 2.

Algorithm 2 Event time derivative for stochastic times automata.

If event α is feasible at x_0: $\Delta_\alpha \leftarrow \frac{dV_\alpha}{d\psi}$

Else, for all other event $\alpha \in \Xi : \Delta_\alpha \leftarrow 0$

```
1: while Queueing network is in execution do
2:     if Event β is observed then
3:         if Event α is activated with lifetime V_α then
4:             Calculate dV_α/dψ using (3.2)
5:             Calculate Δ_α ← Δ_β + dV_α/dψ
6:         end if
7:     end if
8: end while
```

A Sufficient Condition for Unbiasedness

Based on [27], the equality in (3.1) is enforced by the *dominated convergence theorem* if we can determine a finite upper bound R, $\mathbb{E}\{R\} < \infty$ such that,

$$\left| \frac{L(\psi + \Delta\psi) - L(\psi)}{\Delta\psi} \right| \leq R.$$

Assuming that the sample function $L(\lambda, \psi)$ is continuous and differentiable, the generalized mean value theorem asserts that,

$$\left|\frac{L(\psi + \Delta\psi) - L(\psi)}{\Delta\psi}\right| \leq \sup_{\gamma \in [a,b]} \left|\frac{dL(\gamma)}{d\psi}\right|. \tag{3.4}$$

Therefore, by calculating bounds for the sample derivatives $\frac{dL(\lambda,\psi)}{d\psi}$ we can assure that (3.1) is satisfied and the IPA estimates are unbiased.

3.3.2 Sample Function for Throughput and its Derivative

The model proposed in Section 3.2 should be simplified in order to enforce (3.1) and to be able to implement the IPA approach for performance regulation. In order to satisfy Assumptions 1 and 2 it is enough to assume that the service times have an exponential distribution with parameter $\mu_1, \mu_2, \ldots, \mu_R$ for the queues $Q_1, Q_2, \ldots, Q_R$ respectively. We assume that the servers in the infinite server queueing system Q_0 are identical and their think times occur at a rate ρ according to a Poisson process. This means that the think time process at the output of the server Q_0 when the server is hosting a population of n requests from a total of N requests in the network now have the superposition of $(N - n)$ Poisson processes. This is proven to be a Poisson process with parameter $\rho(N - n)$ [59].

Now, let us assume that the queues $Q_1, Q_2, \ldots, Q_R$ have infinite queueing capacity, implement a FCFS queue discipline and that there is a single class of customers. Moreover, by implementing the probabilistic routing defined by $p_1, \ldots, p_R$ in Fig. 3.2 this network belongs to the Jackson-like networks described by Definition 1, and consequently, CC is satisfied.

The satisfaction of CC can be graphically verified for the two-tier server whose state transition diagram is shown in Fig. 3.4. The two numbers defining the states represent the queue length of Q_1 and Q_2 respectively. Based on the model in Fig.

3.2, let us denote by a the event of an arrival to queue Q_1, and by d_{ij} the event of a departure from Q_i to Q_j with $i \in \{1, 2, \ldots, R\}$ and $j \in \{l \in \{0, 1, \ldots, R\} : [(l = i - 1) \vee (l = i + 1)] \text{ for some } i\}$. As an example, for a three-tier server the possible departure events are $d_{10}, d_{12}, d_{21}, d_{23}$ and d_{32}. From Theorem 3, assuming that the system is in a particular initial state x, and if, based on the available events, we follow the state transitions defined by a specific sequence of events, we should be able to reach the same final state by following the same sequence of events but in reversed order, *e.g.,* if we start at state 11, and then we follow the sequence of events $\{a, d_{32}, d_{10}\}$, with transitions indicated by the red arrows in Fig. 3.4, then the system ends up at state 20. If we start again at state 11 and we follow the reversed sequence of events, namely, $\{d_{10}, d_{32}, a\}$, with transitions indicated by the pink arrows, the system ends up at state 20 again, proving that CC is satisfied.

Estimation of Throughput

To continue with our development, let us define $T_{\alpha,M}$ as the time of the M-th occurrence of some event α. The throughput y of the multitier model is measured at the output of the first tier, since a departure d_{10} means a request has been completed. Consequently, the sample function for the throughput of the system after M requests have been completed is given by,

$$y(\mu_1) = \frac{M}{T_{d_{10,M}}}, \tag{3.5}$$

where $T_{d_{10},M}$ is the time of the M-th occurrence of the event d_{10}.

Since we want to estimate the derivative of the throughput y with respect to the mean service rate of the first tier μ_1, we proceed to calculate $\frac{dy}{d\mu_1}$ by applying the chain rule and we get,

$$\frac{dy}{d\mu_1} = \frac{-M}{T^2_{d_{10},M}} \left(\frac{dT_{d_{10},M}}{d\mu_1} \right), \tag{3.6}$$

therefore, all that is left is to estimate $\frac{dT_{d_{10},M}}{d\mu_1}$.

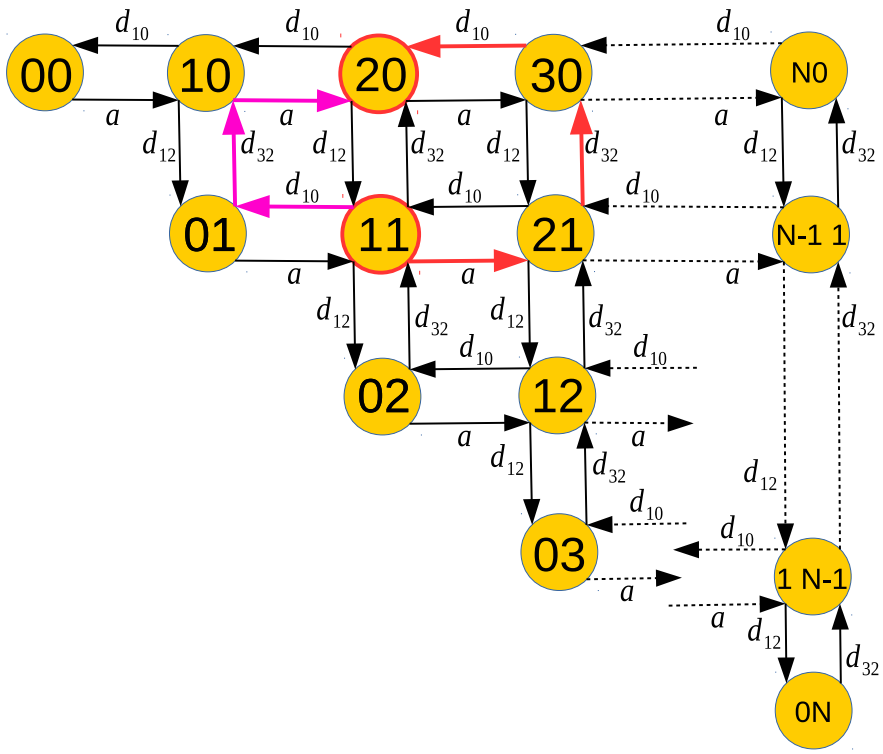

Figure 3.4: State transition diagram of the two-tier server model.

Regulation Algorithm

The regulation algorithm follows the same approach described in Section 2.2.2. This means that given a reference value y_{ref} and with the quadratic error defined by,

$$e^2 = (y_{ref} - y)^2, \tag{3.7}$$

the update of the parameter μ_1 is carried out through the equation,

$$\mu_{1_n} = \mu_{1_{n-1}} + K \frac{e_{n-1}}{\frac{dy(\mu_1)}{d\mu_1}}, \tag{3.8}$$

where $K \in \mathbb{R}$ is a positive scalar that determines the step-size of the gradient descent optimizer given by (3.8) with $n = 1, 2 \ldots$.

Unbiasedness of IPA for Multitier Servers

The following corollary, provides us with the sufficient conditions for continuity of the throughput y.

Corollary 1 *If Assumptions 1 and 2, as well as the CC, are satisfied by the queueing network described in Fig. 3.2, then the sample function,*

$$L_{d_{10},M}(\psi) = \int_0^{T_{d_{10},M}} dt = T_{d_{10},M},$$

is (almost surely) continuous in ψ for finite $T_{d_{10},M}$.

Proof This proof trivially follows from Theorem 11.1 in [27]. □

From Corollary 1, and under the assumptions we have made on the model in Section 3.3.2, we have that y given by (3.5) is continuous. The proof of unbiasedness is out of the scope of this section, but roughly speaking, by taking advantage of the Markovian nature of our simplified model and assuming that the lifetimes $\frac{dV_{d_{10},j}}{d\psi} < c < \infty$, for some $c \in \mathbb{R}$ we should be able to carry out a case-by-case analysis of the states of the network [27]. It turns out that all possible combinations of the perturbation propagation will be upper bounded by polynomial combinations of c and M which are finite. The generalized mean value theorem then implies (3.4), allowing us to apply the dominated convergence theorem, thus the unbiasedness of the IPA estimates is assured.

Estimation of Throughput Derivative for One-tier Servers

Now, we proceed to estimate $\frac{dT_{d_{10},M}}{d\mu_1}$. We apply the generalized Algorithm 2 to the one-tier server model shown in Fig. 3.5. In this problem, the two possible events are the arrival of a request to the server Q_1 denoted by a, and the departure of a

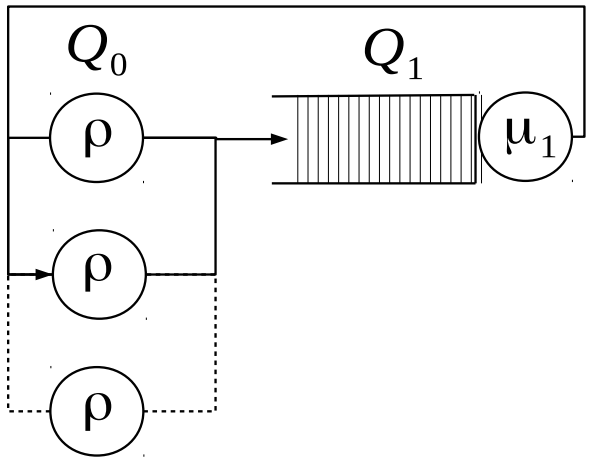

Figure 3.5: Model of a one-tier server.

request from Q_1 denoted by d_{10}. Notice that if the server is not empty, *i.e.,* the queue-length $x_1 \neq 0$, the lifetime of the event d_{10}, namely, $V_{d_{10},i}$, with $i = 1, 2, \ldots, M$ depends only on the $(i-1)$-th occurrence of the same event d_{10}. Remember that the service times are governed only by the parameter μ_1. If the event a is observed while $x_1 \neq 0$, it will not have any effect on the lifetime $V_{d_{10},i}$. However, if the queue is empty, a departure d_{10} will not be feasible until the first arrival a is observed, *i.e.,* if $x_1 = 0$, then the lifetime $V_{d_{10},i}$ depends on the first arrival a. After that, the queue will not be empty, *i.e.,* $x_1 \neq 0$ and the following observation of d_{10} will depend on the observation of the previous departure.

Based on Algorithm 2, we have that if the event a is observed and $x_1 = 0$, the perturbation propagation $\Delta_{d_{10}}$ is obtained by $\Delta_{d_{10}} = \Delta_a + \frac{dV_{d_{10}}}{d\mu_1}$. On the other hand, if $x_1 \neq 0$ and d_{10} is observed, then, the perturbation propagation $\Delta_{d_{10}}$ is obtained by $\Delta_{d_{10}} = \Delta_{d_{10}} + \frac{V_{d_{10}}}{d\mu_1}$ and Δ_a is updated by $\Delta_a = \Delta_{d_{10}} + \frac{dV_a}{d\mu_1} = \Delta_{d_{10}}$. Remember that $\frac{dV_a}{d\mu_1} = 0$ because the think time Y depends on ρ and not on μ_1. This result is illustrated in Algorithm 3.

Algorithm 3 Event time derivative for one-tier servers.

Initial state $x_1 \leftarrow 0$, then event a is feasible and $\Delta_a \leftarrow 0$

and event d_{10} is unfeasible, then $\Delta_{d_{10}} \leftarrow 0$

while Queueing network is in execution **do**
  if Event a is observed $\wedge\ x_1 = 0$ **then**
    $\Delta_{d_{10}} \leftarrow \Delta_a + \frac{dV_{d_{10}}}{d\mu_1}$
  end if
  if Event d_{10} is observed $\wedge\ x_1 \neq 0$ **then**
    $\Delta_a \leftarrow \Delta_{d_{10}}$
    $\Delta_{d_{10}} \leftarrow \Delta_{d_{10}} + \frac{dV_{d_{10}}}{d\mu_1}$
  end if
end while

Remark 1 *In this model, we are assuming that the service times have an exponential distribution with parameter μ_1 which is a scale parameter. The calculation of $\frac{dV_{d_{10}}}{d\mu_1}$ is provided by (3.3), therefore, $\frac{dV_{d_{10}}}{d\mu_1} = \mu_1 Z_1$, where, as established in Section 3.2, Z_1 is the random variable defining the service time of the tier Q_1.*

Remark 2 *The perturbation propagation provides an estimate of the derivative $\frac{dT_{d_{10}}}{d\mu_1}$, given by $\Delta_{d_{10}}$. Hence, based on (3.6), the estimate of the derivative of the throughput at the M-th observation of the event d_{10} is given by,*

$$\hat{y}' = \frac{-M}{T^2_{d_{10},M}} \left(\Delta_{d_{10}}\right). \tag{3.9}$$

Estimation of Throughput Derivative for Three-tier Servers

The model of the three-tier server is shown in Fig. 3.6. This queueing network has the possible events $a, d_{10}, d_{12}, d_{21}, d_{23}$ and d_{32}. One of the main differences with the

one-tier case is the presence of the routing probabilities p_1 and p_2. The effect of the routing probabilities becomes apparent when estimating the event time derivatives. In the three-tier server case, the service times at every tier are assumed to be exponential, and queues Q_1 and Q_2 present bifurcations at the output. These bifurcations have distributions formed by a superposition of exponential random variables with parameter μ_1 and μ_2, and Bernoulli random variables with parameter p_1 and p_2. In particular, the event time derivatives associated with the departure events at tier Q_1 namely, $\frac{dV_{d_{10}}}{d\mu_1}$ and $\frac{dV_{d_{12}}}{d\mu_1}$ are given by,

$$\frac{dV_{d_{10}}}{d\mu_1} = Z_1\mu_1(1-p_1), \tag{3.10}$$

$$\frac{dV_{d_{12}}}{d\mu_1} = Z_1\mu_1 p_1. \tag{3.11}$$

This is equivalent to defining the scale parameter associated with the event d_{10} as $\mu_1(1-p_1)$ and the one associated to d_{12} as $\mu_1 p_1$.

Therefore, the analysis of the dependencies between lifetimes and events should consider the states of each tier. Let us illustrate this idea by studying the perturbation propagation when the parameter μ_1 is changed in tier Q_1. Let us assume that the event d_{12} is observed. Events d_{12} and d_{10} are coupled because both are departures from Q_1 and depend on μ_1. Therefore, by following Algorithm 2, we have that the perturbation $\Delta_{d_{10}}$ is obtained by $\Delta_{d_{10}} = \Delta_{d_{12}} + \frac{dV_{d_{10}}}{d\mu_1}$ and $\Delta_{d_{12}}$ is obtained by $\Delta_{d_{12}} = \Delta_{d_{12}} + \frac{dV_{d_{12}}}{d\mu_1}$. However, if $x_2 = 0$ then Q_2 is receiving a first arrival after an idle time, which means that the events d_{21} and d_{23} are activated. Hence, $\Delta_{d_{21}}$ is updated by $\Delta_{d_{21}} = \Delta_{d_{12}} + \frac{dV_{d_{21}}}{d\mu_1} = \Delta_{d_{12}}$ and $\Delta_{d_{23}}$ is updated by $\Delta_{d_{23}} = \Delta_{d_{12}} + \frac{dV_{d_{23}}}{d\mu_1} = \Delta_{d_{12}}$. Remember that $\frac{dV_{d_{21}}}{d\mu_1} = \frac{dV_{d_{23}}}{d\mu_1} = 0$ because the service time Z_2 depends on μ_2 and not on μ_1. Notice that variations of μ_1 in Q_1 couples with the events d_{21} and d_{23} in tier Q_2 only when the state $x_2 = 0$. Following similar analyses for all the possible events in the system, we obtain Algorithm 5, shown in Appendix A, to estimate $\frac{dT_{d_{10},M}}{d\mu_1}$.

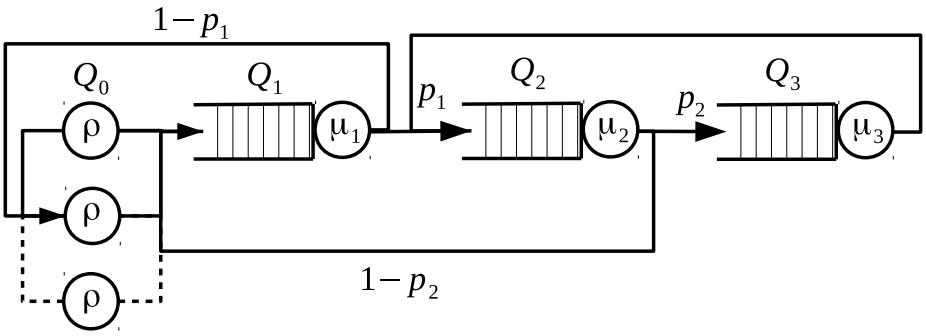

Figure 3.6: Model of a three-tier server.

Remark 3 *The event time derivatives in Algorithm 5, in Appendix A, are calculated using (3.10) and (3.11).*

Remark 4 *Similar to the one-tier server, the perturbation propagation provides an estimate of the derivative* $\frac{dT_{d_{10},M}}{d\mu_1}$*, which is provided by* $\Delta_{d_{10}}$*. Hence, the estimate of the derivative of the throughput at the i-th observation of the event* d_{10} *is given by (3.9).*

3.4 Statistical Learning for Optimal Parameterization

Up to this point, we have been able to guarantee the regulation of throughput to a reference value y_{ref}. However, we are interested not only in regulating throughput but also in reducing power consumption. The problem of throughput regulation

is solvable for infinite configurations of μ_1, μ_2 and μ_3. Roughly speaking, since we are assuming $M/M/1$ queues whose arrivals are supplied by Poisson processes, the departure processes of the queues in the network will be Poisson distributed by *Burke's Theorem* [27]. This implies that for queues Q_1, Q_2 and Q_3, their respective throughputs, namely, y_1, y_2 and y_3 will be given by,

$$y_1 = \begin{cases} \lambda(N-n) & \text{if } \mu_1 > \lambda(N-n) \\ \mu_1 & \text{otherwise} \end{cases}, \tag{3.12}$$

$$y_2 = \begin{cases} \mu_1 p_1 & \text{if } \mu_2 > \mu_1 p_1 \\ \mu_2 & \text{otherwise} \end{cases}, \tag{3.13}$$

$$y_3 = \begin{cases} \mu_2 p_2 & \text{if } \mu_3 > \mu_2 p_2 \\ \mu_3 & \text{otherwise} \end{cases}. \tag{3.14}$$

From (3.12)–(3.14), we can infer that as long as the mean arrival rate parameters, namely, $\lambda(N-n), \mu_1 p_1$ and $\mu_2 p_2$, are less than or equal to their correspondent mean service rate parameters, namely, μ_1, μ_2 and μ_3 respectively, we should be able to regulate the throughput of all three queues by controlling μ_1. Otherwise, the throughputs are saturated by a value proportional to one of the mean service rates μ_i with $i = 1, 2, 3$. This in turn, implies that there is not a unique triplet (μ_1, μ_2, μ_3) to regulate the throughput to a certain value. In fact, the set of possible triplets that provide a solution is infinite. Therefore, we propose an optimization approach to reduce the values of the triplet of mean service rate while guaranteeing the regulation of the throughput of the system.

This optimization step is carried out in open-loop based on power requirements. We take advantage of the direct relation between power and throughput making use of DVFS. As mentioned in Section 2, the consumed power is directly proportional to the frequency of operation and this, in turn, is directly proportional to the mean service rate μ_i. The main goal is to minimize the mean service rates μ_i so that the

regulation error (3.7) is minimized on average. This will provide a reduced power consumption while guaranteeing a minimum regulation error in the system.

3.4.1 Performance Function

Let us assume that we have a set of R tiers in a multitier processor. The throughput of the system, namely, y is given by (3.5) which, as explained above, depends on the output of the tier Q_1.

Let us define the parameter set,

$$\boldsymbol{\Psi} = \left\{ \boldsymbol{\psi} \in \mathbb{R}^R : \boldsymbol{\psi} = (\mu_1, \ldots, \mu_R) \right\},$$

which consists of the vector containing the mean service rates of all tiers.

Based on hardware considerations we determine the finite bounds for the parameters μ_k, such that, $\mu_k \in [\mu_{k_{\min}}, \mu_{k_{\max}}]$, with $k = 1, 2, \ldots, R$. Thus, we define, $\bar{\mu}_k = \frac{\mu_k - \mu_{k_{\min}}}{\mu_{k_{\max}} - \mu_{k_{\min}}}$, so that $\bar{\mu}_k \in [0, 1]$. Similarly, let us assume that we can calculate finite bounds for e in (3.7), such that $e \in [e_{\min}, e_{\max}]$, so we define $\bar{e} = \frac{e - e_{\min}}{e_{\max} - e_{\min}}$.

Now, let us define the random sample,

$$\lambda = y \in \boldsymbol{\Lambda} \subset \mathbb{R},$$

and the performance function,

$$J(\lambda, \boldsymbol{\psi}) = J = \alpha_k \sum_{k=1}^{R} \bar{\mu}_k + \beta \bar{e}, \tag{3.15}$$

where $\alpha_k, \beta \in \mathbb{R}^+$ are chosen so that $J : \boldsymbol{\Lambda} \times \boldsymbol{\Psi} \to [0, 1]$.

3.5 Simulation Results

To validate our approach we carried out a simulation of a queueing network modeling the three-tier server. For this simulation we assume a set of $N = 100$ concurrent requests in the system, a mean think time $\frac{1}{\rho} = 33\frac{1}{3}$ s and nominal probabilities $p_1 = p_2 = \frac{1}{2}$. Our goal is to regulate the throughput of the system to a reference value $y_{ref} = 3$ requests/s while minimizing the mean service rates $\mu_1, \mu_2, \mu_3 \in [1, 50] \subset \mathbb{R}$ with accuracy $\epsilon = 0.02$ and confidence $1 - \delta = 0.98$. We have chosen $\alpha_k = \frac{1}{30}$ with $i = 1, 2, 3$ and $\beta = 0.9$ in (3.15). Furthermore $K = 0.1$ in (3.8).

For the given values of ϵ and δ and using Algorithm 1 we obtain that $M_1 = 228$ values of the parameter triplet (μ_1, μ_2, μ_3) should be evaluated. Furthermore, per each evaluated parameter triplet, $M_2 = 13,410$ samples should be observed. After running Algorithm 1 the system calculates the following mean service rates $\mu_1^* = 2.4262$, $\mu_2^* = 8.3696$, and $\mu_3^* = 28.1239$ request/s.

In Fig. 3.7 we show the transient behavior of the regulated throughput. The blue line with circular markers shows the regulation applied to the model with routing probabilities $p_1 = p_2 = 0.5$, *i.e.,* the same used for the optimization. The green line with triangular markers shows the regulation when the routing probabilities are perturbed so that their value changed to $p_1 = 0.6$ and $p_2 = 0.4$. Notice that even in the perturbed case the system manages to stay regulated. However, regulating the perturbed system requires a higher mean service rate μ_1 as illustrated in Fig. 3.8, which implies more power consumption in tier Q_1. The regulation error is illustrated in Fig. 3.9, which, in accordance with Fig 3.7, goes asymptotically to zero.

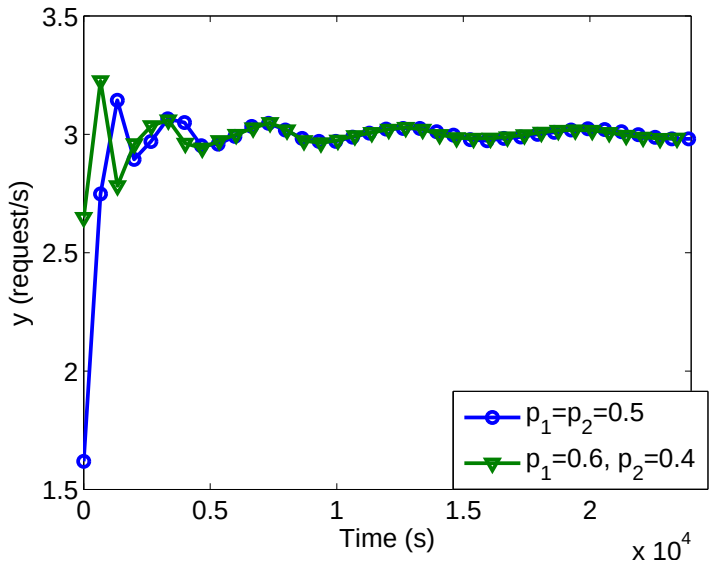

Figure 3.7: Plot of regulated throughput for a three-tier server.

3.6 Conclusions

We have presented an approach that optimizes and regulates performance in a multitier server. In this particular case, given a reference value for the throughput, we optimize the values of the mean service rates at each tier in the server. This parameter is directly proportional to the operation frequencies of the tiers, and therefore, by using a DVFS approach we are able to guarantee power consumption reduction while satisfying throughput requirements. Statistical learning is used to calculate reduced values of the mean service rates of the multiple tiers present in the system that satisfy the throughput requirements. After the optimization process is carried out, an IPA algorithm is implemented to regulate the throughput of the system to the reference value in a closed loop. The control parameter of the throughput corresponds to the mean service rate μ_1 of Q_1. In consequence, the values of the mean service rates of all but the front-end tier Q_1 are kept constant in their original value, while μ_1 keeps

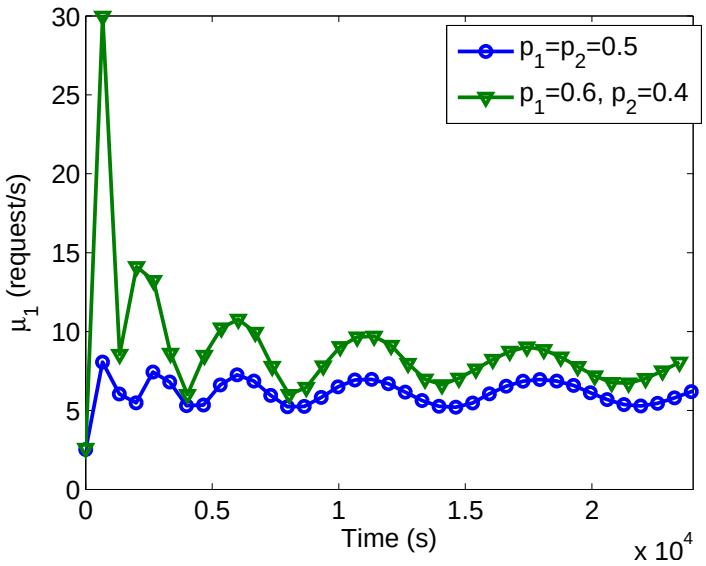

Figure 3.8: Plot of the controlled parameter μ_1 for a three-tier server.

the general throughput of the system regulated. We validate our approach using a queueing network model that satisfies the required sufficient conditions to guarantee unbiased IPA estimates.

We validated our results by carrying out the simulation of a three tier server in Matlab®. The results showed that the system is able to calculate the optimal mean service rate parameters in all three servers for the given accuracy and confidence. The IPA-based regulator not only converges to the reference value of throughput, but keeps it regulated under small variations of the routing probabilities.

Although the initial simulations results show that the system carried out the reduction of power and the regulation of throughput, the mathematical simplifications of the model does not allow the analysis of more sophisticated and complex control mechanisms, such as, contention control, load balancing or processing sharing that cover more realistic scenarios. Since recent evidence has revealed that biased

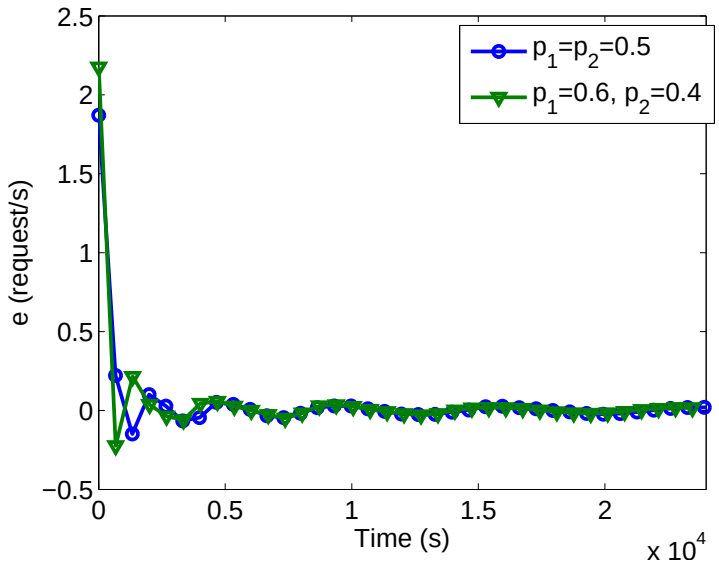

Figure 3.9: Plot of the regulation error of the throughput for a three-tier server.

and bounded IPA estimates may be useful in regulation and optimization problems, our future research will explore models of more complex queueing networks in order to cover several of the aforementioned control mechanisms and include them in the optimization problem.

Chapter 4

Cloud Computing Model with Time-Varying Workload

4.1 Introduction

In the last few years, control theory has had a productive but still limited relationship with computing theory and systems [15, 16]. Control theory is being used in problems such as managing power consumption for microprocessors [17], data centers [19, 20], application performance [60, 61] and management of resources in cloud computing [19, 22].

At the cloud computing level, and within the infrastructure-as-a-service (IaaS) framework [62], the costumer controls the software running over a virtual server which has been instantiated by a resource provider. Resources are usually leased and may consist of application and storage servers. Currently available services include AWS EC2, Google Cloud and Joyent. In some services the users must rely on coarse-grained visibility of the system [24, 62]. A common control theory-based approach involves model identification and optimal control [6, 62] where, under certain

assumptions a model of the cloud is estimated in order to control and optimize some performance measure, *e.g.,* latency and throughput. The problem of virtual resource allocation to regulate application performance may then be studied as discussed in [6, 24, 25].

In the recent paper [1], the author proposes a passivity framework to ensure asymptotic stability of a feedback controlled system where the controller regulates power while guaranteeing response time management in the cloud. The author proposes a market-oriented discrete-time model to describe the routing of the consumer's workload in the cloud through the interaction of *brokers* and *servers*. Roughly speaking, the servers communicate with the brokers to let them know how busy they are, while the brokers distribute the consumer's workload to be processed between the servers based on the current status of the servers. This framework takes advantage of the passivity inherited by a system formed by interconnecting passive subsystems.

Following the ideas proposed by [1] we present a number of enhancements related to the analysis of the market-oriented cloud model. Among other results, we provide mathematical propositions to justify the use of passivity theory to the analysis of this problem, and an additional sufficient condition to guarantee the asymptotic stability of the system. Furthermore, we provide comments about the stability of the system in the presence of time-varying consumer's workload.

This chapter is organized as follows: Section 4.2 describes the market-oriented cloud model as presented in [1]. In Section 4.3 we present passivity analysis to study the asymptotic stability of the system. Section 4.4 analyzes a counter-example to illustrate the need for an additional sufficient condition for asymptotic stability. Such condition is explicitly provided and proven. In Section 4.5, we prove that the system is robust to time-varying consumer's workload as long as such workload is bounded. In Section 4.6 we present simulation results to validate our approach. Finally, in Section 4.7 we provide our conclusions.

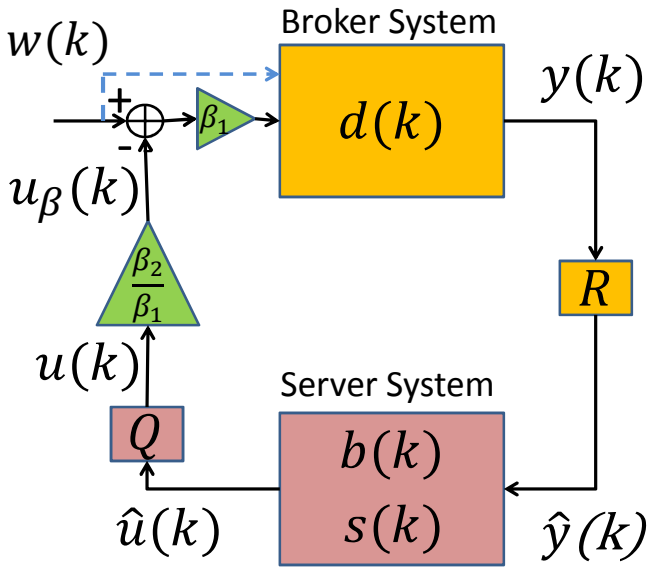

Figure 4.1: Block diagram of the market-oriented cloud presented in [1].

4.2 Market-Oriented Cloud Model

The discrete-time model for the cloud based on a market-oriented view was proposed in [1]. The block diagram in Fig. 4.1, illustrates the dynamics corresponding to a set $\mathcal{B}$ of N brokers and a set $\mathcal{S}$ of M servers. A consumer's workload vector $\mathbf{w}(k) \in \mathbb{R}^N$ serves as a reference input for the system at time k. The amount of workload that should be routed to the servers is calculated by the set of brokers. The set of brokers send the vector $\mathbf{y}(k) \in \mathbb{R}^N$ with the dispatched workload and the servers receive a fraction of the consumer's workload through the vector $\hat{\mathbf{y}}(k) \in \mathbb{R}^M$. The fraction of the workload that is not completed is buffered and the servers send a throttling signal vector $\hat{\mathbf{u}}(k) \in \mathbb{R}^M$ that tells the brokers the current load at the servers. Thus, the brokers receive the vector signal $\mathbf{u}(k) \in \mathbb{R}^N$ and, based on it, calculate the fraction of $\mathbf{w}(k)$ that should be routed to the servers in the next time iteration.

The following state-space equations for the j-th broker were proposed in [1]. The state $d_j(k)$ with $j = 1, 2, \ldots, N$, corresponds to the *maximum dispatch level* at time instant $k \in \mathbb{Z}^+$, and the dynamics are given by:

$$d_j(k+1) = \left[(1-\beta_{1j})d_j(k) + \beta_{1j}w_j(k) - \beta_{2j}u_j(k)\right]^+, \tag{4.1}$$

with $\beta_{1j} \in (0,1) \subset \mathbb{R}$ and $\beta_{2j}(0,\infty) \subset \mathbb{R}$. The projection operator is defined as $[\cdot]^+ = \max(\cdot, 0)$ and the output is given by,

$$y_j(k) = \min\{w_j(k), d_j(k)\}. \tag{4.2}$$

Similarly, the state-space model of the i-th server is defined by two state variables, namely, $b_i(k)$ and $s_i(k)$ with $i = 1, 2, \ldots M$. $b_i(k)$ corresponds to the amount of pending workload to be processed by the i-th server. $s_i(k)$ corresponds to the maximum amount of workload that the i-th server processes at time instant k and is upper bounded by the physical limit service $\bar{s}_i$. Thus, the state-space equations are given by,

$$b_i(k+1) = \left[b_i(k) + \hat{y}_i(k) - s_i(k)\right]^+, \tag{4.3}$$

$$s_i(k+1) = \min\{\bar{s}_i, (1-\sigma_i)s_i(k) + b_i(k) + \hat{y}_i(k)\}, \tag{4.4}$$

with $\sigma_i \in (0,1)$. The designed output in [1] is given by,

$$\hat{u}_i(k) = 2\sigma_i b_i(k) + 2\sigma_i s_i(k). \tag{4.5}$$

This system may be expressed in matrix form as,

$$\xi_i(k+1) = \min\{\bar{\xi}_i, [\mathbf{A}_i\xi_i(k) + \mathbf{B}_i\hat{y}_i(k)]^+\}, \tag{4.6}$$

$$\hat{\mathbf{u}}_i(k) = \mathbf{C}\xi_i(k). \tag{4.7}$$

where,

$$\xi_i(k) = \begin{pmatrix} b_i(k) \\ s_i(k) \end{pmatrix}, \quad \bar{\xi}_i = \begin{pmatrix} \infty \\ \bar{s}_i \end{pmatrix},$$

$$\mathbf{A}_i = \begin{pmatrix} 1 & 1 \\ \sigma_i & 1-\sigma_i \end{pmatrix},$$

$$\mathbf{B}_i = \begin{pmatrix} 1 \\ \sigma_i \end{pmatrix}, \mathbf{C} = (2\sigma_i, 2\sigma_i).$$

The broker and server blocks are connected through the matrices $\mathbf{R}(k) \in \mathbb{R}^{M\times N}$ and $\mathbf{Q}(k) \in \mathbb{R}^{N\times M}$. Such matrices allow for the consideration of a different number of brokers and servers in the model. The entries of the matrices, namely, $R_{ij}(k)$ and $Q_{ji}(k)$ satisfy $\sum_i R_{ij}(k) = 1$ and $\sum_j Q_{ji}(k) = 1$.

From now on, we omit the subindices j and i when referencing the j-th and i-th entries of the corresponding vectors in the systems of brokers and servers. To analyze the passivity of the j-th broker, [1] proposes the following storage function,

$$V_1(d) = d^2(k),$$

and assuming that the reference input of the system $w(k) = 0$ and the projection in (4.1) is inactive we have that

$$\Delta V_1 \leq u(k)d(k) + ((1-\beta_1)^2 - 1)d(k),$$

which indicates that the system is output strictly passive. Notice that $((1-\beta_1)^2 - 1)d^2 \leq 0$ since $\beta_1 \in (0, 1)$.

By assuming that the projection in (4.1) is active, the difference of the storage function becomes,

$$\Delta V_1 = -d^2(k) \leq u(k)d(k) - d^2(k),$$

and [1] concludes that the broker system is output strictly passive.

Analyzing the passivity of the server system, [1] also proposes the following storage function,

$$V_2(\xi) = \xi^T(k)\mathbf{P}\xi(k), \tag{4.8}$$

with the positive definite matrix,

$$\mathbf{P} = \begin{pmatrix} \sigma & -\sigma/2 \\ -\sigma/2 & 1 \end{pmatrix}.$$

The projection in (4.3) is assumed to be inactive and the first difference of (4.8) satisfies the following inequality,

$$\Delta V_2 \leq \hat{u}(k)\hat{y}(k).$$

Similarly, using the same storage function but assuming that the projection in (4.3) is active, [1] concludes that

$$\Delta V_2 \leq \hat{u}(k)\hat{y}(k),$$

and the system is shown to be passive. Therefore, from Proposition 1 and Proposition 2 in [1] the origin of the feedback system with $w(k) = 0$ is asymptotically stable. One interesting result of this approach is the utilization of passivity concepts to calculate a *certificate* that once satisfied, guarantees the stable operation of the system.

4.3 More About Passivity Analysis

Even though the passivity approach was already applied to this problem, it is worth asking the following question: Is it possible to apply passivity analysis to the market-oriented cloud system? The answer is yes, but we must be careful. Recall that the

output of the j-th broker is $y(k) = \min\{d(k), w(k)\}$, then $\mathbf{w}(k)$ is an input of the broker system as shown in Fig. 4.1. When carrying out the passivity analysis in [1], the reference input is assumed to be $w(k) = 0$ then $y(k) = 0$.

Since $y(k) = d(k)$ if and only if $d(k) \leq w(k)$, special care should be taken before directly applying the passivity propositions to show asymptotic stability as presented in [1]. However, if we are able to prove that there exists a finite number of time steps $N \in \mathbb{Z}^+$ such that $y(k) = d(k), \forall k \geq k_0 + N$, we can eliminate the input $w(k)$ indicated by the dashed blue arrow in Fig. 4.1, and make sure that $y(k) = d(k)$ as described next.

Proposition 2 *Consider the state-space dynamics of the j-th broker defined by (4.1) and (4.2). For any initial condition $d(k_0)$ such that $d(k_0) > w(k) = w > 0$ with $w \in \mathbb{R}^+$ constant, there exists $N < \infty$, $N \in \mathbb{Z}^+$ such that*

$$y(k) = \min\{w, d(k)\} = d(k), \qquad \forall k \geq k_0 + N.$$

Proof Let us define a new state variable

$$d_0(k) = d(k) - w(k) + \frac{\beta_2}{\beta_1} u(k), \tag{4.9}$$

therefore, we obtain the new dynamical equation,

$$d_0(k+1) = (1 - \beta_1) d_0(k), \ \beta_1 \in (0, 1). \tag{4.10}$$

Now, let us propose the following Lyapunov function candidate,

$$V_3(d_0) = d_0^2(k),$$

and the first difference gives,

$$\Delta V_3 = (-1 + (1 - \beta_1)^2) d_0^2(k) \leq 0, \tag{4.11}$$

and the origin of (4.10) is asymptotically stable.

Let us assume an initial condition $d(k_0) > w$. Furthermore, from (4.11) we know that for any $d(k_0) > 0$ there exists $\eta \in \mathbb{R}^+$ such that

$$\Delta V_3 < (-1 + (1 - \beta_1)^2) d_0^2(k) < -\eta,$$

therefore,

$$V_3(k+1) - V_3(k) < -\eta,$$

and solving the recurrence equation we get,

$$V_3(k) \leq V_3(k_0) - (k - k_0)\eta, \tag{4.12}$$

If we take any feasible $\delta \in \mathbb{R}^+$ in the trajectory of $d_0(k)$ such that $d_0(k_0) > \delta > 0$ we get $V_3(\delta) = \delta^2$. Therefore, if starting from the initial condition $d_0(k_0)$ we arrive at $d_0(k) = \delta$ for some k, we get from (4.12) that,

$$\delta^2 \leq V_3(d(k_0)) - (k - k_0)\eta,$$

therefore,

$$k \leq k_0 + \frac{V_3(k_0) - \delta^2}{\eta} < \infty.$$

Then, the number of steps required to go from any initial state $d_0(k_0) > 0$ to another state $d_0(k) > 0$ in the trajectory of the solution of (4.10) is finite. From (4.9) we conclude that starting from an initial state $d(k_0)$, there exists $N \in \mathbb{Z}^+, N < \infty$ such that $0 < d(k) \leq w,\ \forall k > k_0 + N$, therefore, $y(k) = \min\{w, d(k)\} = d(k) > 0,\ \forall k > k_0 + N$. □

Remark 5 *Notice that for the case $w(k) = w = 0$ the foregoing Proposition does not apply, since from [1], $d(k) \to 0$ as $k \to \infty$ asymptotically,* i.e., *in infinite time.*

Remark 6 *Notice that in Proposition 2 we do not consider the case where the projection of $d(k)$ is active because we have assumed that $d(k) > 0$.*

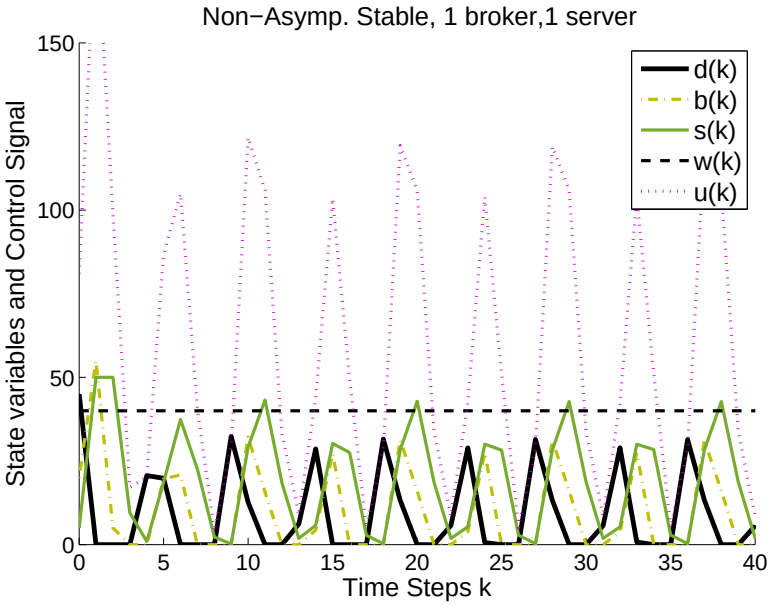

Figure 4.2: Non-asymptotically stable example that satisfies the sufficient conditions for asymptotic stability given in [1].

4.4 Effect of Equilibrium Points in Stability

As mentioned before, based on [1], the market-oriented cloud described in Section 4.2 was shown to be asymptotically stable. However, let us implement the foregoing model assuming only one broker and one server with $\beta_1 = 0.9$, $\beta_2 = 0.9$, $w(k) = w = 40$, $\bar{s} = 50$ and initial conditions $d(k_0) = 45, b(k_0) = 20, s(k_0) = 5$. Furthermore, let us replace (4.5) by,

$$\hat{u}_1(k) = 2\sigma b(k) + 2\sigma s(k) + \sigma \hat{y}(k). \tag{4.13}$$

with $\sigma = 0.9$. We provide a detailed justification for using (4.13) in Appendix B.

We obtain the plot shown in Fig. 4.2, which does not show an asymptotically stable trajectory. In order to explain the result, we provide the following Proposition,

Proposition 3 *A condition to guarantee the asymptotic stability of the feedback connection between the broker system given by (4.1) and (4.2), and the server system given by (4.3), (4.4) and (4.13) is,*

$$0 < \beta_2 u(k) \leq \beta_1 w(k). \tag{4.14}$$

Proof let us calculate the equilibrium points of the j-th broker and the j-th server,

$$d_{eq} = w(k) - \frac{\beta_2}{\beta_1} u(k), \tag{4.15}$$

$$b_{eq} = 0,$$

$$s_{eq} = y(k). \tag{4.16}$$

Now, let us assume

$$0 < w(k) < \frac{\beta_2}{\beta_1} u(k), \tag{4.17}$$

but from the projection in (4.1) the equilibrium point $d_{eq} \geq 0$, which contradicts (4.17). Therefore,

$$w(k) \geq \frac{\beta_2}{\beta_1} u(k) \geq 0, \tag{4.18}$$

and the sufficient condition (4.14) follows. □

Examining the plots of $u_\beta(k) = \frac{\beta_2}{\beta_1} u(k) = u(k)$ and $w(k)$ in Fig. 4.2, we see that at every oscillation, the inequality (4.18) is not satisfied at some time intervals, therefore, asymptotic stability cannot be guaranteed.

4.5 Effect of Time-Varying $\mathbf{w}(k)$

Now, we consider our last question: Do bounded inputs guarantee bounded states in the market-oriented model? This property is termed *input-to-state stability* (ISS) and

is related to the capacity of the states of the system to remain in the neighborhood of its equilibrium points. In this specific case, we are assuming that $\mathbf{w}(k)$ in (4.1) is a time-varying vector function. Although the simulation results shown in [1] suggest that the system may be ISS, this needs to be formally proven as described in the following result.

Proposition 4 *Given the feedback connection in Fig. 4.1 defined by the broker system with dynamics (4.1) and (4.2) with* $0 < \beta_1 < 1,\ \ \beta_1 \in \mathbb{R}$ *and* $0 < \beta_2,\ \beta_2 \in \mathbb{R}$*, and the server system with dynamics (4.3),(4.4) and (4.13) with* $\sigma \in (0, 1) \subset \mathbb{R}$*, the resulting system is ISS. Furthermore, if* $0 < \beta_2 u(k) < \beta_2 w(k)$*, the system tracks the equilibrium points (4.15)–(4.16) asymptotically.*

Proof Let us study the broker system defined by (4.1) and (4.2), and let us define,

$$e_1 = w(k) - \frac{\beta_2}{\beta_1} u(k),$$

then, the system (4.1) may be rewritten as,

$$d(k+1) = \left[(1 - \beta_1) d(k) + \beta_1 e_1\right]^+ . \tag{4.19}$$

Assuming no active projection in (4.19), we notice that the system is *linear time invariant* (LTI). It was proven in [1] that the broker system is output-strictly passive with a positive definite storage function. Since it is zero-state observable as well, the origin with $e_1(k) = 0$ is asymptotically stable. Since the system is LTI we conclude that it is *bounded-input-bounded-output* (BIBO) stable as well.

Similarly, the server system given by (4.3), (4.4) and (4.13) was proven to be passive with a positive definite storage function, and therefore its origin is stable with $\hat{y}(k) = 0$. Assuming no active projection, the dynamics are given by (4.6) and (4.7) which describe an LTI system, therefore the system is BIBO stable. Assuming

an active projection in the server system the system matrix $\mathbf{A}$ becomes,

$$\mathbf{A} = \begin{pmatrix} 0 & 0 \\ \sigma_i & 1-\sigma_i \end{pmatrix},$$

and the system is LTI, therefore it is BIBO stable.

Furthermore, with $y(k) < \infty$ and $\hat{\mathbf{y}}(k) = \mathbf{R}(k)\mathbf{y}(k)$ where all the entries of $\mathbf{R}(k)$, namely, $R_{ij}(k) \in [0,1]$, then $\hat{y}(k) < \infty$. Since $\mathbf{u}(k) = \mathbf{Q}(k)\hat{\mathbf{u}}_1(k)$ where the entries of $\mathbf{Q}(k)$, namely, $Q_{ji}(k) \in [0,1]$, then $u(k) < \infty$.

Since we know that $\frac{\beta_2}{\beta_1}u(k) < \infty$ and $w(k) < \infty$, then $w(k) - \frac{\beta_2}{\beta_2}u(k) < \infty$, and the system is ISS. If in addition (4.14) is satisfied, then the system tracks the equilibrium points asymptotically. □

4.6 Simulation Results

In Fig. 4.3, we present simulation results using two brokers and three servers with $\beta_1 = 0.95$, $\beta_2 = 0.1$, $\sigma = 0.5$, $\bar{s} = 20$ and $w(k) = 12.5 + 12.5\sin(\frac{2\pi k}{100})$. Notice that (4.18) is being fulfilled, since we are plotting $u_{1\beta}(k) = \frac{\beta_2}{\beta_1}u(k)$ represented by the purple dotted line, which avoids undamped oscillating behaviors. Notice that all the other states are in the neighborhood of their respective equilibrium points given by (4.15)–(4.16). Notice also that the subindices in the plot of Fig. 4.3, indicate that we are plotting the inputs and outputs of broker 1 and server 2 respectively.

Furthermore, now that we have shown that the system is ISS stable, we are able to assure that the states remain bounded, as long as the consumer's workload stays bounded. In comparison with the results shown in [1] we carry out simulations using three brokers and five servers. The parameters are $\beta_1 = 0.95$, $\beta_2 = 0.1$, $\sigma = 0.5$ and $\bar{s} = 20$. The consumer's workload is modeled as a Gaussian white noise with mean $\mu = 2$ and variance $\sigma_g^2 = 1$. At time step $k = 100$ the mean of $w(k)$ abruptly

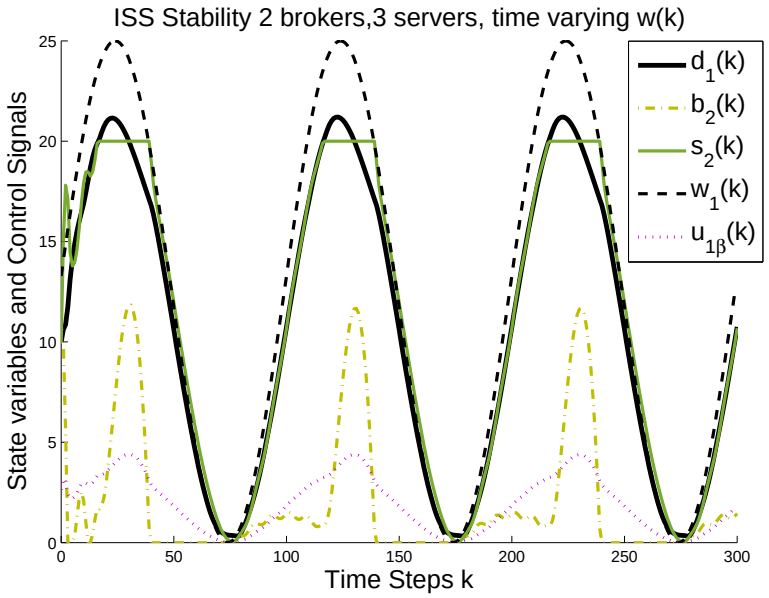

Figure 4.3: Simulation showing that the market-oriented cloud model in [1] is ISS.

changes to $\mu = 25$ and at $k = 210$ it goes back to $\mu = 2$. Later, at $k = 425$ the mean goes to $\mu = 16$ and at $k = 632$ it returns to $\mu = 2$. As anticipated by the theory, the system is ISS. In Fig. 4.4, 4.5 and 4.6 we observe that all states remain bounded. However, in contrast to the simulation presented in Fig. 4.3, we can no longer assure the asymptotic tracking of the equilibrium points (4.15)–(4.16) because the natural oscillations due to the stochastic nature of the process do not guarantee that condition (4.14) is satisfied.

4.7 Conclusions

We have presented an in depth analysis of the passivity framework introduced in [1] for power control and response time management in the cloud. We enhanced the original theoretical result with a detailed analysis of the stability and stabilization

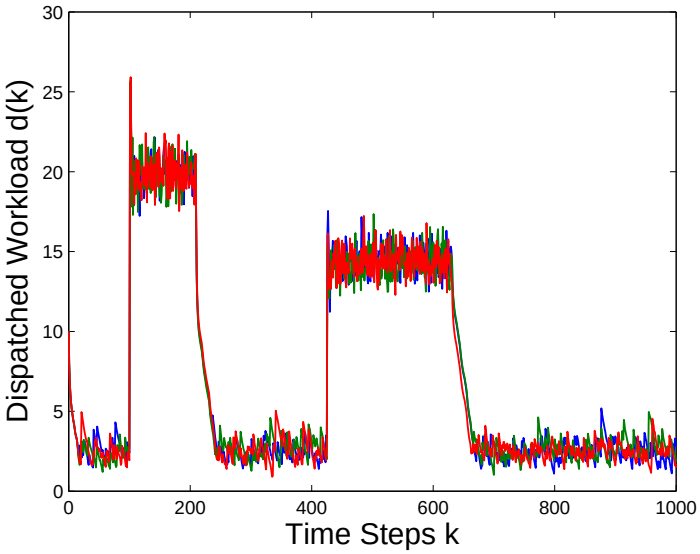

Figure 4.4: Simulation showing bounded $d(k)$ with random and bounded $w(k)$.

of the system. We have presented a rigorous approach to guarantee that passivity analysis is suitable for this specific problem in order to guarantee asymptotic stability. Moreover, using a counterexample as a starting point, we have formally provided an additional sufficient condition for the asymptotic stability of the market oriented cloud model. Furthermore, we have formally proven that the proposed cloud model is ISS in the presence of time-varying consumer's workload vectors. All the theoretical results have been validated through simulation.

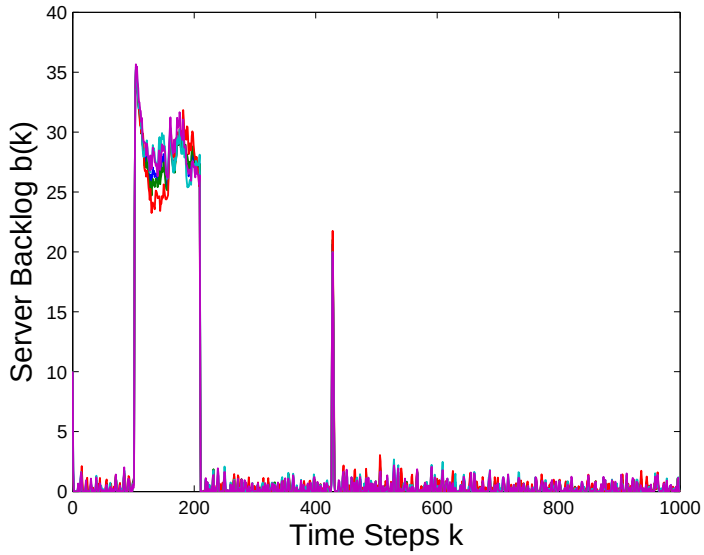

Figure 4.5: Simulation showing bounded $b(k)$ with random and bounded $w(k)$.

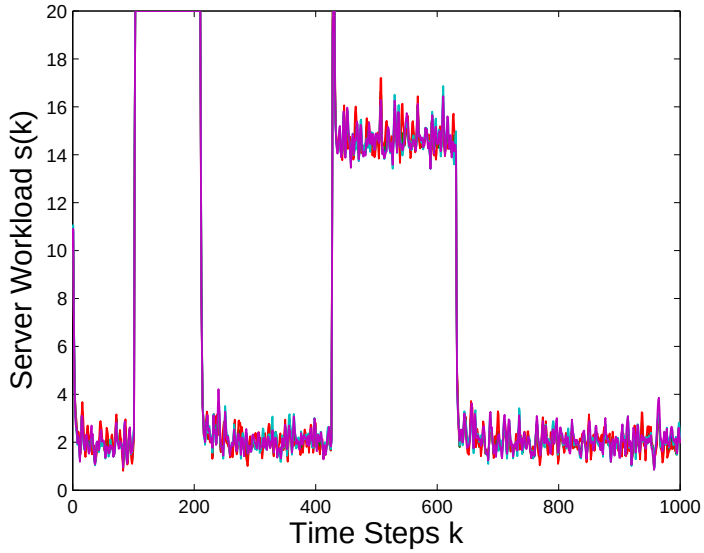

Figure 4.6: Simulation showing bounded $s(k)$ with random and bounded $w(k)$.

Chapter 5

Resource and Security Provisioning on the Cloud

5.1 Introduction

The negotiation of SLAs in IaaS in the cloud remains a challenging problem. One of the main difficulties of guaranteeing performance in the cloud is the inherent randomness produced by the massive amount of time-varying interactions and events taking place in the system. However, it is possible to optimize different performance measures on the average based on the amount and size of the available virtual resources. This optimization process may be complemented with some closed-loop control methodologies related to *control systems theory* and DES [23, 26, 28] to regulate the calculated optimal performance. Although many approaches have been proposed to overcome this problem, commercial clouds have not been able to implement systems where users pay for specific performance measures such as CPU and memory utilization rather than a flat hourly rate service.

In commercial IaaS, such as the one provided by AWS EC2, the input variables

are coarse-grained [62, 63]. This fact complicates the application of performance regulation and optimization using multivariable control systems theory as in the CPU controller proposed in [64] or the performance regulation of the cloud in [6]. In those two cases, the main assumption is that the input variables of the system are fine-grained. In [62], the authors present the *proportional thresholding* technique which is effective in keeping performance confined in an interval for systems with coarse-grained inputs. However, this approach is limited to *single-input-single-output* (SISO) systems. Another methodology implements online model identification [23, 26, 65] by using well known techniques such as, *recursive least square* (RLS) or *least mean square* (LMS) filters [26] to approximate the dynamics of the cloud to a linear system and then to find a regulating controller. These techniques are however prone to oscillations whenever the input variables are coarse-grained.

Given the complexities involved in the definition of security, as well as its dependency on particular concepts and applications, the formulation of metrics of security is a difficult problem. In the work presented in [13, 14], the authors proposed quantitative methods to measure cloud security levels based on *Reference Evaluation Methodology* (REM) and *Quantitative Policy Trees*. This methods, in the IaaS context, allow for the incorporation of security in the SLA. However, this is something that has not been formally proposed yet. Moreover, there is a proven need to protect sensitive information stored in and traveling through the cloud. Encryption offers a solution to some of the current security issues [66, 67] associated with data storage.

In order to address all of these issues, we present a probabilistic method for resource optimization and security provisioning in the cloud based on RAs [31]. We propose a multi-objective function which allows the addition of metrics of security based on cryptographic algorithms. To the best of the authors' knowledge this is the first formal approach that incorporates security along performance as a negotiable variable in the SLA. This chapter is organized as follows: In Section 5.2 we introduce

a sample complexity result for finite families. This result support our probabilistic approach for optimization. In Section 5.3 we present an example of a metric of security based on cryptographic ciphers and provide the details of the multi-objective function. Moreover, we explain the RA for optimization and describe a complementary heuristic algorithm to reduce the execution time of the optimization process. In Section 5.4 we present an implementation using the AWS EC2 service and validate our approach through experimental results. In Section 5.5 we present our conclusions.

5.2 Probabilistic Performance Analysis

As suggested above, our approach points towards a technique that allows us to optimize the distribution of large amounts of virtual resources in the cloud among a large number of users or clients. One of the main tools supporting this technique are *tail inequalities* which consist of closed mathematical expressions that bound the probability that random variables with no compact support take values in the tail of the distribution, *i.e.,* far from the mean [59]. Markov's and Chebychev's inequalities are typical examples of such inequalities. Next, we introduce mathematical results to support the implementation of our probabilistic resource optimization.

5.2.1 Worst-case Performance for Finite Families

In this section, we present the calculation of the sample complexity for *worst-case performance* in finite families, but first, we introduce the following result from [31].

Corollary 2 *Given the performance function* $J : \mathbf{\Lambda} \to \mathbb{R}$ *and the multi-sample* $\boldsymbol{\lambda}^{(1,\ldots,N_1)} \subseteq \mathbf{\Lambda}$ *picked at random, let us define the constant* $\gamma_{N_{1_{\min}}} = \min_{i=1,\ldots,N_1} J(\boldsymbol{\lambda}^{(i)})$.

For any $\delta \in (0,1)$ *and* $\epsilon \in (0,1)$*, if*

$$N_1 \geq \frac{\ln \frac{1}{\delta}}{\ln \frac{1}{1-\epsilon}}, \tag{5.1}$$

then,

$$P_R\{J(\boldsymbol{\lambda}) \geq \gamma_{N_{1_{\min}}}\} \geq 1-\epsilon,$$

with probability of at least $1-\delta$*.*

Proof The proof is provided in [68].

The sample complexity for worst-case performance in Corollary 2 assumes a single performance function. The following result applies whenever we need to optimize over a finite family of performance functions.

Lemma 1 *Given the performance function* $J: \boldsymbol{\Lambda} \times \boldsymbol{\Psi} \rightarrow \mathbb{R}$*, where* $\boldsymbol{\Psi}$ *is a finite set of parameter vectors with cardinality* $\tilde{n}_C \leq n_C$ *and the multi-sample* $\boldsymbol{\lambda}^{(1,\ldots,N_2)} \subseteq \boldsymbol{\Lambda}$ *picked at random, let us define the finite family* $\mathcal{J}_{\tilde{n}_C} = \left\{J(\boldsymbol{\lambda}, \boldsymbol{\psi}^{(1)}), \ldots, J(\boldsymbol{\lambda}, \boldsymbol{\psi}^{(\tilde{n}_C)})\right\}$ *and the constant* $\gamma_{N_{2_{\min}}} = \min_{i=1,\ldots,N_2} J(\boldsymbol{\lambda}^{(i)}),\ \forall J \in \mathcal{J}_{\tilde{n}_C}$*.*

For any $\delta_2 \in (0,1)$ *and* $\epsilon_2 \in (0,1)$*, if*

$$N_2 \geq \frac{\ln \frac{n_C}{\delta_2}}{\ln \frac{1}{1-\epsilon_2}}, \tag{5.2}$$

then,

$$P_R\left\{\forall J \in \mathcal{J}_{n_C} : J(\boldsymbol{\lambda}) \geq \gamma_{N_{2_{\min}}}\right\} \geq 1-\epsilon_2,$$

with confidence $1-\delta_2$*.*

Proof From the proof of Corollary 2 in [68], we get,

$$P_R\left\{P_R\{J(\boldsymbol{\lambda}) \geq \gamma_{N_{2_{\min}}}\} \geq 1-\epsilon_2\right\} > 1-(1-\epsilon_2)^{N_2},$$

therefore,

$$P_R\left\{P_R\left\{J(\boldsymbol{\lambda}) < \gamma_{N_{2_{\min}}}\right\} > \epsilon_2\right\} < (1-\epsilon_2)^{N_2}.$$

Now, we proceed to bound the probability of deviation for all $J \in \mathcal{J}_{\tilde{n}_C}$,

$$P_R\left\{\exists J \in \mathcal{J}_{\tilde{n}_C} : P_R\left\{J(\boldsymbol{\lambda}) < \gamma_{N_{2_{\min}}}\right\} > \epsilon_2\right\}$$

$$\begin{aligned} &\leq \sum_{i=1}^{\tilde{n}_C} P_R\left\{P_R\left\{J(\boldsymbol{\lambda}) < \gamma_{N_{2_{\min}}}\right\} > \epsilon_2\right\} \\ &< \sum_{i=1}^{\tilde{n}_C} (1-\epsilon_2)^{N_2} \leq n_C(1-\epsilon_2)^{N_2}. \end{aligned}$$

Therefore, $n_C(1-\epsilon_2)^{N_2} \leq \delta_2$ should be satisfied, and solving the inequality for N_2 we obtain (5.2).

Remark 7 *Since the minimum value of N_2 is directly proportional to $\ln n_C$, this approach becomes more efficient for large values of n_C.*

Remark 8 *Note that for small values of δ_2 and ϵ_2 it may happen that $n_C < N_2$,* i.e., *we get more samples to draw than parameters $\boldsymbol{\psi} \in \boldsymbol{\Psi}$. This does not represent a contradiction, since the random samples are taken with replacement for our particular problem. The cloud is a very complex and interconnected system [69] with permanent variations of performance, therefore, there are no guarantees of observing the same behavior when running the same test in two instances with the same configuration running the same benchmarks.*

Our performance optimization and security provisioning problem aims at minimizing a cost function J to be described in detail in the sections to follow. Given the

complexities of the IaaS environment in the cloud, we propose an RA to carry out such minimization through the calculation of a probable minimum of $\hat{\mathbb{E}}_N(J(\boldsymbol{\lambda}, \psi))$. To fulfill this goal, we are required to determine the required sample complexity to solve the optimization problem with a given accuracy and confidence. This aspect is addressed by the following corollary.

Corollary 3 *Given the empirical probable parameter vector,*

$$\hat{\boldsymbol{\psi}}_{M_1 M_2} = \arg \min_{i=1,\ldots,M_2} \hat{\mathbb{E}}_{M_1}(J(\boldsymbol{\lambda}, \boldsymbol{\psi}^{(i)})),$$

and the performance function $J : \boldsymbol{\Lambda} \times \boldsymbol{\Psi} \rightarrow [0, 1]$*, where* $\boldsymbol{\Psi}$ *is a finite set of parameter vectors with cardinality* $\tilde{n}_C \leq n_C$*. Let,*

$$M_2 \geq \frac{\ln \frac{2n_C}{\delta}}{\ln \frac{1}{1-\epsilon_2}}, \tag{5.3}$$

and

$$M_1 \geq \frac{\ln \frac{4M_2}{\delta}}{2\epsilon_1^2}, \tag{5.4}$$

then,

$$P_R \left\{ \mathbb{E}\left(J(\boldsymbol{\lambda}, \boldsymbol{\psi})\right) < \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda}, \boldsymbol{\psi}_{M_1 M_2})\right) - \epsilon_1 \right\} \leq \epsilon_2,$$

with probability at least $1 - \delta$*.*

Proof Let us define $\delta = 2\delta_1 = 2\delta_2$ in (5.3) and (5.4), then by Lemma 1 we are guaranteed that,

$$\left| \mathbb{E}\left(J(\boldsymbol{\lambda}, \psi)\right) - \hat{\mathbb{E}}_{M_1}(J(\boldsymbol{\lambda}, \psi)) \right| \leq \epsilon_1 \tag{5.5}$$

and

$$P_R \left\{ \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda}, \boldsymbol{\psi})\right) \geq \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda}, \hat{\boldsymbol{\psi}}_{M_1 M_2})\right) \right\} \geq 1 - \epsilon_2, \tag{5.6}$$

hold with joint confidence of at least $(1-\frac{\delta}{2})^2 > (1-\delta)$. Furthermore, (5.6) implies that,

$$P_R\left\{\hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi})\right) < \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda},\hat{\boldsymbol{\psi}}_{M_1M_2})\right)\right\} < \epsilon_2.$$

From (5.5) we get that,

$$\hat{\mathbb{E}}_{M_1}(J(\boldsymbol{\lambda},\boldsymbol{\psi})) \leq \mathbb{E}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi})\right) + \epsilon_1,$$

therefore,

$$P_R\left\{\mathbb{E}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi})\right) + \epsilon_1 < \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi}_{M_1M_2})\right)\right\}$$

$$\begin{aligned} &\leq P_R\left\{\hat{\mathbb{E}}_{M_1}(J(\boldsymbol{\lambda},\boldsymbol{\psi}) < \hat{\mathbb{E}}_{M_1}\left(J(\boldsymbol{\lambda},\boldsymbol{\psi}_{M_1M_2})\right)\right\} \\ &< \epsilon_2. \end{aligned}$$

□

5.3 Resource Optimization in the Cloud

As mentioned before, previous research considered the optimal distribution of virtual resources in the cloud [6, 69, 70]. Although security metrics have been proposed in the past [13, 14], never before has security been formally introduced as a resource to be optimized and provided along with other performance measures. In this methodology, we not only address the challenging problem to serve SLAs based on performance rather than on amount of resources, but we also provide a mathematical framework to incorporate security as a service in the cloud. The following sections provide the details of this technique.

5.3.1 Security Metrics based on Cryptography

Securing information stored in the cloud is a crucial problem. Given the amount of variables associated with the very complex concept of security, the process of determining a metric for the "amount" of security is far from being straightforward and univocal. For this specific problem we propose assigning values to different security levels as shown in Table 5.1. We restrict the concept of security to the organization and implementation of several cryptographic algorithms or ciphers for data storage. We are aware that security encompasses a broader set of techniques and methodologies, and other security metrics may later be incorporated in this approach. Moreover, Table 5.1 can be easily modified to satisfy the particular needs of the cloud service providers and the clients.

In the first column of Table 5.1 we present the security measure values given by numbers between 0 and 1. Security performance increases going from top to bottom. This is not only based on the key sizes, but on the ability of the cryptographic algorithm to be run in parallel, to be synchronizable, and its immunity to cryptanalysis. As described later in Section 5.3.2, the security level is the only variable inversely proportional to the performance function in this particular approach. In the second column we describe the cipher, the key size and its mode of operation by using various security standards. The highest value of 1 is assigned to the *No encryption* option. The next security level value goes to the *data encryption standard* (DES[1]). After that, we follow with the *advanced encryption standard* (AES) with the *electronic codebook* (ECB) mode of operation and a key size of 128 bits. Following the sequence, we proceed to increase the key size up to 256 bits, and then we proceed to add enhancements by progressively changing the modes of operation which go from *cipher-block chaining* (CBC), *cipher feedback* (CFB), cipher feedback with shift registers (CFB-1/CFB-8), *output feedback* (OFB) and *counter* (CTR). All

[1]Not to be confused with the initial of *Discrete Event Systems* used in previous sections.

Table 5.1: Measure of security associated to ciphers and modes of operation

Sec. Lvl $\overline{S}_i$	**Cipher Mode of Op.**	**Observation**
1	No encryption	No encryption whatsoever
0.39984	DES	Short key sizes, 64-bit encryption blocks, time issues with large files, prone to cryptanalysis
0.15978	AES-128-ECB	Does not hide data patterns well, 128-bit key size
0.06375	AES-192-ECB	Does not hide data patterns well, 192-bit key size
0.02534	AES-256-ECB	Does not hide data patterns well, 256-bit key size
0.00998	AES-256-CBC	Non-parallel encryption, 256-bit key size
0.00383	AES-256-CFB	Non-parallel encryption, 256-bit key size, No padding
0.00138	AES-256-CFB-1 AES-256-CFB-8	Non-parallel encryption, 256-bit key size, Synchronizable, No padding
0.00039	AES-256-OFB	Non-parallel encryption, 256-bit key size, Synchronizable, faster block cipher operations
0	AES-256-CTR	Parallel encryption, 256-bit key size, Synchronizable

the relevant advantages and disadvantages of each mode of operation are specified in the third column of the table. Notice that the ciphers found in the table can be subject to cryptanalysis and, after a successful attack, may be ruled out for use in sensitive applications. Therefore, one of the advantages of this approach is that by using tables such as Table 5.1, we are able to change them and update them ac-

cording to the security requirements of our problem. Moreover, the security metrics presented in [13, 14] are compatible with this methodology.

5.3.2 RA for Optimization

One of the goals of our approach is to calculate the optimal distribution of resources among different users of IaaS while offering different levels of security. As discussed earlier, the inherent randomness of the behavior of the cloud complicates the search for an optimal solution. The ever increasing number of potential users and available virtual resources motivates the implementation of a decentralized methodology.

Within the scope of SLA requirements between users and cloud service providers, we propose to optimize the distribution of cloud resources based on the user needs, while guaranteeing cost savings to the provider. In this section, an unconstrained optimization problem is proposed based on the mathematical framework provided in Section 5.2.

Performance Function

Let us assume that we have a set of $\tilde{N}$ users whose resources need to be optimized. For the k-th user, we consider the following performance metrics for optimization,

$$\begin{aligned} C_{\mu_k} &= \% \text{ of CPU utilization,} \\ M_{\mu_k} &= \% \text{ of memory utilization,} \\ T_k &= \text{Total execution time of benchmark,} \\ W_k &= \text{Hourly cost of instance usage.} \end{aligned}$$

with $k = 1, \ldots, \tilde{N}$.

Let us define the parameter set,

$$\begin{aligned}\boldsymbol{\Psi} \quad = \quad & \{\text{inst-type-1}, \text{inst-type-2}, \ldots, \text{inst-type-}n_1, \\ & \text{enc-cipher-1}, \text{enc-cipher-2}, \ldots, \text{enc-cipher-}n_2, \\ & \text{volume-1}, \text{volume-2}, \ldots, \text{volume-}n_3\}.\end{aligned}$$

which encompasses the set of instance types, security levels and volume (hard drive) sizes available to the users to configure the required virtual resources. n_1, n_2 and n_3 represent the number of instances, volume sizes and security levels available to the users respectively.

With the possible exception of the processing time $T \geq 0$, all the foregoing variables are upper and lower bounded by finite real numbers. It is reasonable to assume an upper bound for T that may be statistically estimated by an RA for probabilistic worst-case performance [31]. By defining upper and lower bounds for the random variables, we are able to calculate the normalized versions, $\overline{C}_{\mu_k}, \overline{M}_{\mu_k}, \overline{T}_k, \overline{W}_k \in [0, 1] \subset \mathbb{R}$.

Next, let us define the random vector,

$$\boldsymbol{\lambda}_k = \left(\overline{C}_{\mu_k}, \overline{M}_{\mu_k}, \overline{T}_k, \overline{W}_k\right)^T \in \boldsymbol{\Lambda},$$

and propose the following performance function $J_k(\boldsymbol{\lambda}, \boldsymbol{\psi}) = J_k$ for the k-th user,

$$J_k = \frac{1}{5}\mathbb{E}\left\{\alpha_{1_k}\overline{C}_{\mu_k} + \alpha_{2_k}\overline{M}_{\mu_k} + \alpha_{3_k}\overline{T}_k + \alpha_{4_k}\overline{S}_k + \alpha_{5_k}\overline{W}_k\right\}, \tag{5.7}$$

then, we define the performance vector function,

$$\mathbf{J} = \left(J_1, \ldots, J_N\right)^T, \tag{5.8}$$

where $\alpha_{1_k}, \ldots, \alpha_{5_k} \in [0, 1]$ correspond to the weights given to the variables based on the SLA requirements of the k-th client in the cloud.

Remark 9 *The proposed optimization method aims at maximizing security while minimizing the remaining metrics in (5.7) at the users' convenience. This implies that the minimization of the security metric should decrease as the security performance improves. This goes according to the organization of the metrics in Table 5.1.*

Remark 10 *Note that the minimization of* $\overline{C}_{\mu_k}$ *does not imply the minimization of* $\overline{M}_{\mu_k}$*, since some benchmark problems that affect CPU performance do not necessarily affect memory usage and vice-versa.*

Remark 11 *It is expected that* $\overline{C}_{\mu_k}$ *and* $\overline{M}_{\mu_k}$ *are inversely proportional to* $\overline{T}_k$*, so the weights* $\alpha_{j_k}, j = 1 \dots 5$*, would determine whether minimizing* $\overline{C}_{\mu_k}$ *and* $\overline{M}_{\mu_k}$ *is more important than minimizing* $\overline{T}_k$ *based on the user's needs.*

The RA to optimize performance and provision security is described in Algorithm 4. It is entirely based on Corollary 3.

Algorithm 4 Performance and Security Optimization

Define: $J : \mathbf{\Lambda} \times \mathbf{\Psi} \to [0, 1]$

Calculate: $n(\mathbf{\Psi}) \leftarrow \tilde{n}_C, \quad \tilde{n}_C \leq n_C$ ▷ Cardinality of $\boldsymbol{\psi}$

Define: $\epsilon_1, \epsilon_2, \delta \in (0, 1)$

1: $M_2 \leftarrow \frac{\ln \frac{2n_C}{\delta}}{\ln \frac{1}{1-\epsilon_2}}$ ▷ According to Corollary 3

2: $M_1 \leftarrow \frac{\ln \frac{4M_2}{\delta}}{2\epsilon_1^2}$

3: $\boldsymbol{\psi} \leftarrow \text{randSamples}(M_1)$ ▷ Draw M_1 samples of $\boldsymbol{\psi}$

4: $\boldsymbol{\lambda} \leftarrow \text{randSamples}(M_2)$ ▷ Draw M_2 samples of $\boldsymbol{\lambda}$

5: **return** $\hat{\boldsymbol{\psi}}_{M_1 M_2} \leftarrow \arg \min_{i=1,\dots,M_2} \frac{1}{M_1} \sum_{k=0}^{M_1} J(\boldsymbol{\lambda}^{(k)}, \boldsymbol{\psi}^{(i)})$

5.3.3 Heuristics for Execution Time Reduction

One of the main issues related to the implementation of optimization algorithms for virtual resources is the boot time of the instances. Very often, when carrying out hardware configurations for performance tests in the cloud, the instances should be stopped and restarted, *e.g.,* to attach and detach volumes.

Moreover, in order to gather performance measurements, such as CPU and memory usage, we need to have access to system information files. The acquisition and processing of these files produce an additional workload on the system, *i.e.,* by measuring the performance of the instances in the cloud, we are affecting the same performance we are trying to measure. In order to reduce the perturbations in the performance of the system, the measurements should be taken at a low frequency rate. Taking the number of samples given by (5.3) and (5.4) may take prolonged times at the typical sample frequency of 1 Hz adopted by most system monitor tools [71, 72], as the accuracy and confidence requirements become more stringent.

Given the aforementioned prolonged implementation times, we propose a heuristic algorithm to reduce the duration of the optimization. The idea behind this approach consists of taking performance measures in parallel based on the availability of the resources and their combinations. Let us illustrate its operation through an example.

Example: Execution Time Reduction

Let us assume that one user whose virtual resources are to be tested has three different instance types, three different volume sizes and ten security levels available. Every instance, volume and security levels are indexed by integer numbers starting from zero, *e.g.,* the three volume sizes are indexed by $i = 0, 1, 2$. Let us assume that we need to test seven random samples, and after we draw them we get the following sequence $\{020, 120, 120, 219, 210, 200, 107\}$.

The three numbers at each value encode the index of the three available resources, *i.e.,* 219 means that the instance type with index 2, the volume size with index 1 and the security level with index 9 are the instance configuration selected to be tested. Furthermore, notice that if the values in the random sequence are evaluated one at a time and in the order they appear, we can make the following observations:

1. To test each instance configuration we need at least 7 cycles of the algorithm.

2. During the first change of configuration which goes from 020 to 120 in the random sequence, volume 2 which is associated with instance 0 should be detached and attached to instance 1. With the current infrastructure of cloud services, this is only possible by stopping the instance before the detachment is carried out. The time to reboot may take up to several minutes and, based on the random sequence in the example, the instances should reboot three more times.

3. Note that the security level depends on the software configuration of the instance and not on the hardware, therefore, the change of security levels does not determine the reboot of instances.

After the random sequence is generated, Algorithm 6 in Appendix C organizes the samples in the matrix-like disposition illustrated in Fig. 5.1, and which we refer to as *configuration matrix* from now on. The index of each position corresponds to the indexes of the instance type and the volume size, *e.g.,* the numbers 210 and 219 are located in the position 21 of the matrix. The index of the security levels is not considered in the configuration matrix because, as pointed out before, it does not affect the hardware configuration of the instance.

The algorithm explores the entries of the matrix one row at a time from left to right and from top to bottom. Therefore, the first configuration to be tested would

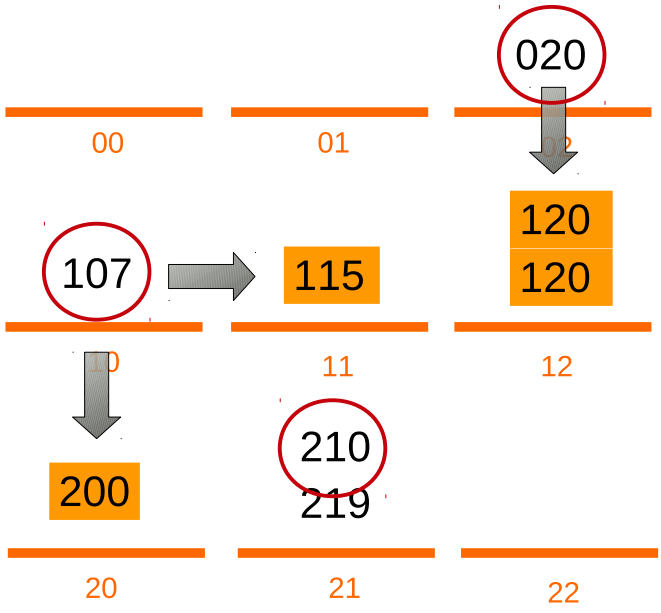

Figure 5.1: Example step 1. The available configurations 020, 107 and 210 (encircled) block the selection of the adjacent remaining entries (highlighted).

be 020. Since the instance type 0 and the volume size 2 have been assigned for testing, then all the entries in row 0 and in column 2 cannot be selected as shown in Fig. 5.1. Continuing with the exploration of the matrix we find that the first entry that is available is the one with index 10, hence, the configuration 107 is selected for testing. This choice automatically blocks the selection of all remaining entries located in row 1 and in column 0. Searching available configurations we find that the entry with index 21 is available, therefore, configuration 210 is chosen. Note that configuration 219 cannot be used along with 210, however, since they share exactly the same resources, switching between both tests would not require instance reboot.

Now, we are ready to compute the configurations to be assessed in the next cycle. First of all, we proceed to discard the configurations that have been tested as indicated by the entries that appear crossed out in Fig. 5.2. Therefore, we search for

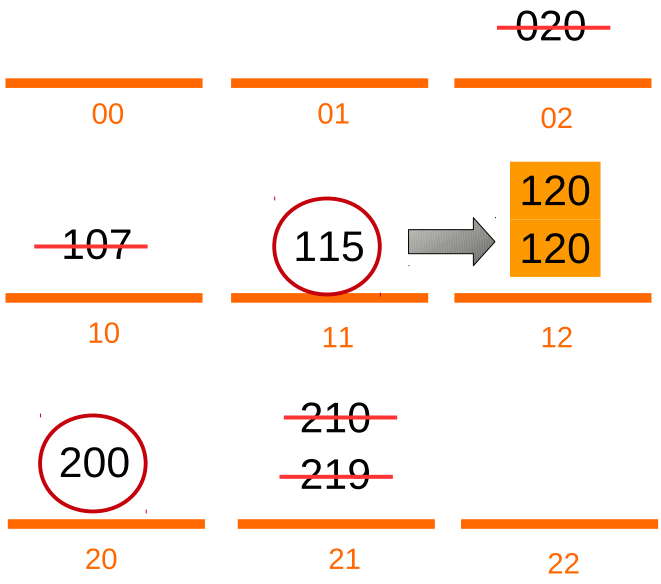

Figure 5.2: Example step 2. The previously selected configurations are discarded (crossed out) and the new available ones are 115 and 200.

an available configuration which turns out to be the entry with index 11, therefore, the selected configuration is 115. This selection blocks the entry whose index is 12. After that, the entry with index 20 is available and is selected to be assessed in the third cycle of our RA.

Finally, we proceed to discard the instance configurations that have been previously chosen for evaluation as indicated in Fig. 5.3. For the fourth cycle of our algorithm, the only instance configuration available is the one in the entry indexed 12, namely, 120 which should be tested twice. Note that instance reboot is not required this time either.

The sequence of configurations to be tested by the RA is shown in Table 5.2. By using the heuristic algorithm, we have reduced the number of cycles from 7 to 5, and the number of instance reboots from 4 to 2. As it will be illustrated in the

experimental section, as the number of available resources increases, this algorithm becomes more efficient in reaching reductions of the required number of cycles of up to 63.91%. The pseudo-code of the heuristic algorithm for execution time reduction is shown in Algorithm 6 in Appendix C.

Table 5.2: Example: Number of cycles of Algorithm 6 and Instance Configurations

Cycle	**Instance Configurations**
1:	020 107 210
2:	219
Reboot Instances	
3:	115 200
Reboot Instances	
4:	120
5:	120

5.4 Experimental Verification

To verify our approach we proceed to carry out experiments by using the AWS EC2 service. We emulate three simultaneous users who have different requirements for IaaS. Each user may choose between five different instance types, namely, *t2.micro*, *t2.small*, *t2.medium*, *m3.medium* and *m3.large*. The technical specifications of each instance can be found in [73]. Furthermore, each user has access to the set of encryption ciphers listed on Table 5.1 and to ten volume sizes containing the root partition of the instances, namely, 12 GB, 14 GB, 16 GB, ..., 30 GB.

To test the performance of each instance, every volume should be previously configured to run their required benchmarks. To emulate this environment we have

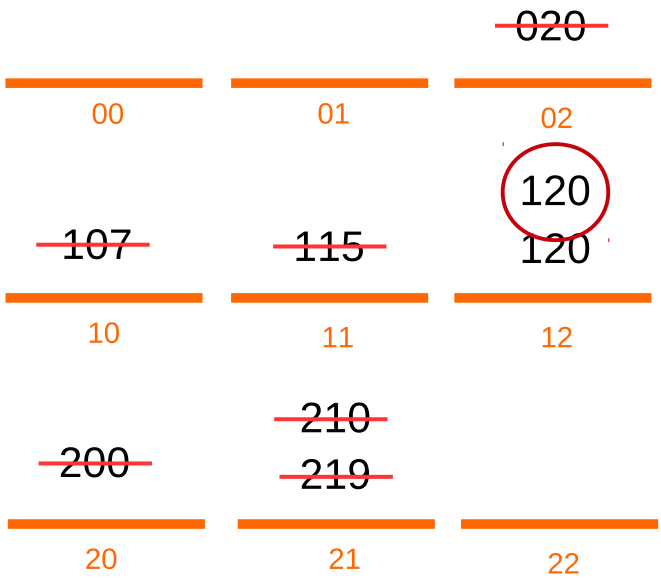

Figure 5.3: Example step 3. The previously selected configurations are discarded and the repeated configuration 120 is the last available one.

prepared an AWS EC2 *snapshot* with an installation of *Ubuntu Server 14.04 LTS*, 64 bits. The system is provided with an installation of the *Java SE Runtime Environment* to be able to run the *dacapo benchmark suite* [74]. This suite incorporates a set of applications with non-trivial memory loads oriented to benchmarking. Since the encryption process affects the general performance of the instance as well, we propose to test security and performance at the same time. This is carried out by encrypting a large file, in this particular case a set of DVD movies (4.7 GB) that are previously stored in the snapshot. The security levels require the installation of OpenSSL [75] which has a full-strength general purpose cryptography library. All the cryptographic ciphers listed on Table 5.1 are available in such a library. Thus, while the system is carrying out the evaluation of performance by running the benchmark problem, the system would be encrypting the DVD files with the selected cipher, providing the desired level of security and the required measurements of performance

listed in Section 5.3.2.

Every instance is in charge of measuring its own performance by running a script developed in Python [76]. Such a script accesses two system files, namely, */proc/stat* to calculate the CPU utilization and */proc/meminfo* to calculate memory utilization. These measurements are taken according to Corollary 3 at a frequency of 1 Hz and then sent to text files. The execution time of the benchmark is extracted from the output of the dacapo benchmark suite. Once all this information is formatted by the python script the probabilistic resource optimization is carried out.

Up to this point, we have assumed that instances, volumes and security levels are somehow configured to work together and form a virtual resource able to run all the applications for performance measurement. However, every client should be able to test different configurations and get measurements. This is carried out through the *Boto SDK* for AWS [77], which provides APIs with capacities for launching a variety of instances, creating customized volumes based on preconfigured snapshots and allowing the attachment of volumes to instances as required, while exploiting the benefits of Python scripts and libraries.

5.4.1 Experimental Results

In this section, we provide experimental results involving three users with different performance requirements. For our first experiment, we carry out the optimization of (5.7) and (5.8) assuming that the k-th users do not have concerns about neither security $\overline{S}_k$ nor hourly cost $\overline{W}_k$ with $k = 1, 2, 3$. Table 5.3 illustrates the priority coefficients for all three users.

Table 5.3: Coefficients for Multi-objective Function in Experiments

Coefficients	**User**$_1$	**User**$_2$	**User**$_3$
Experiment # 1			
	$(k=1)$	$(k=2)$	$(k=3)$
α_{1_k}	1	0	0
α_{2_k}	0	1	0
α_{3_k}	0	0	1
α_{4_k}	0	0	0
α_{5_k}	0	0	0
Experiment # 2			
α_{1_k}	1	0	0
α_{2_k}	0	1	0
α_{3_k}	0	0	1
α_{4_k}	0.05	0.05	0.05
α_{5_k}	0	0	0
Experiment # 3			
α_{1_k}	1	0	0
α_{2_k}	0	1	0
α_{3_k}	0	0	1
α_{4_k}	0.05	0.05	0.05
α_{5_k}	0.1	0.1	0.05

Experiment # 1

From Table 5.3 and from (5.7), we conclude that for User$_1$ the priority is exclusively the minimization of the CPU utilization. For User$_2$ what matters is to minimize the memory utilization. Finally, User$_3$ aims at the minimization of the execution time of the benchmark problem.

Based on Corollary 3, given accuracy $\epsilon_1 = \epsilon_2 = 0.05$ and confidence $1 - \delta = 0.95$

we get $M_1 = 1,930$ and $M_2 = 194$. With our heuristic algorithm the number of cycles is reduced from 194 to 77, a reduction of 60.3%, with only 11 reboots of the tested instances during the entire experiment. Since the sampling rate of the performance is 1 Hz, then, by neglecting the reboot times, the total experiment lasted approximately 41.28 hr.

Based on Fig. 5.4, User_1 (dotted green line) is the only one whose priority is to minimize $\overline{C}_{\mu_1}$, hence exhibiting the best CPU utilization in the plot. Memory utilization is illustrated in Fig 5.5, which according to Table 5.3 is optimized only by User_2 (dashed blue line). Note that since User_2 assigns no priority to the storage security the instance does not carry out file encryption. User_3 (solid red line) is the only one for whom the execution time of the benchmark is the priority. Note from Fig. 5.4 and 5.5 that the red line abruptly decays at $474s$ indicating that User_3 is able to finish the benchmark problem first, however, at a high cost of the CPU and memory utilization. Finally, the optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ with $k = 1, 2, 3$ are shown in Fig 5.6.

Experiment # 2

Our second experiment includes the level of security $\overline{S}_k$ of the instances. A weight of $\alpha_{4_k} = 0.05$ has been added to the performance function of each user. As illustrated in Fig. 5.7 and 5.8, the results are consistent with Experiment # 1, *i.e.,* User_1 continues to get the best CPU performance, User_2 obtains the best average memory usage and User_3 gets the best execution time. However, User_2 and User_3 have changed from *No encryption* to a couple of intermediate encryption ciphers, namely, *AES-256-CFB1* and *AES-256-ECB* as indicated in the plot legend. This is coherent with the increase in security that was requested by the user. In this case, the heuristic algorithm reduced the number of cycles from 194 to 72, a reduction of 62.88%. The number of total instance reboots was 13. The plot of the performance functions for

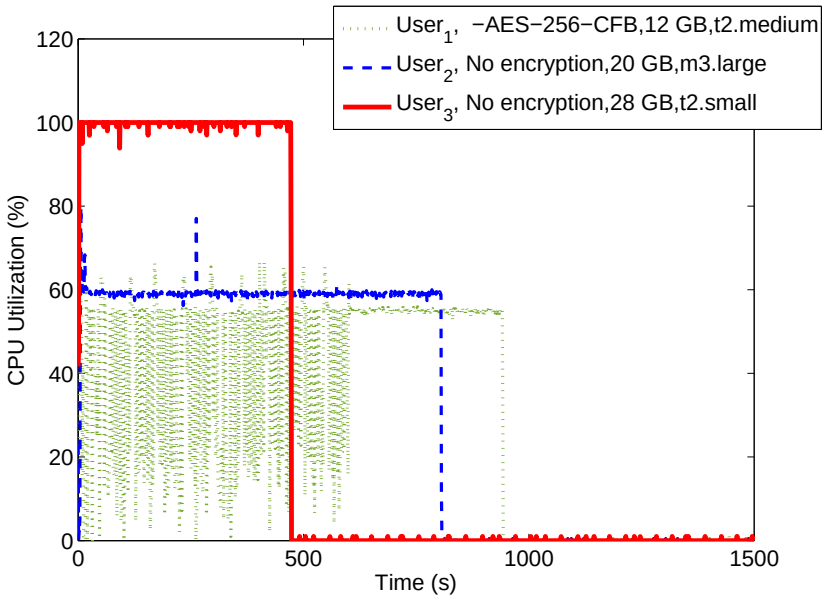

Figure 5.4: CPU utilization for all three users in Experiment # 1.

all three users is shown in Fig. 5.9.

Experiment # 3

Finally, we assume that all users decided to incorporate the hourly cost of the instances in the optimization problem. Therefore, the weights shown in Table 5.3 have been added to the performance function. From Fig. 5.10 and 5.11 we get that the inclusion of these coefficients was enough to change the instances initially used by $User_2$ from *m3.large* to *t2.medium*. $User_3$ changed from instance *t2.small* to *t2.micro*. In both cases, the hourly cost was reduced as expected. However, the variables considered in the performance function can be conflictive, and as expected, the cost reduction of the instances affects the CPU and memory performance. This is clearly visible in Fig. 5.11, where in spite of the good performance of $User_2$'s average memory usage in the experiment, it seems to be surpassed by $User_1$'s performance after

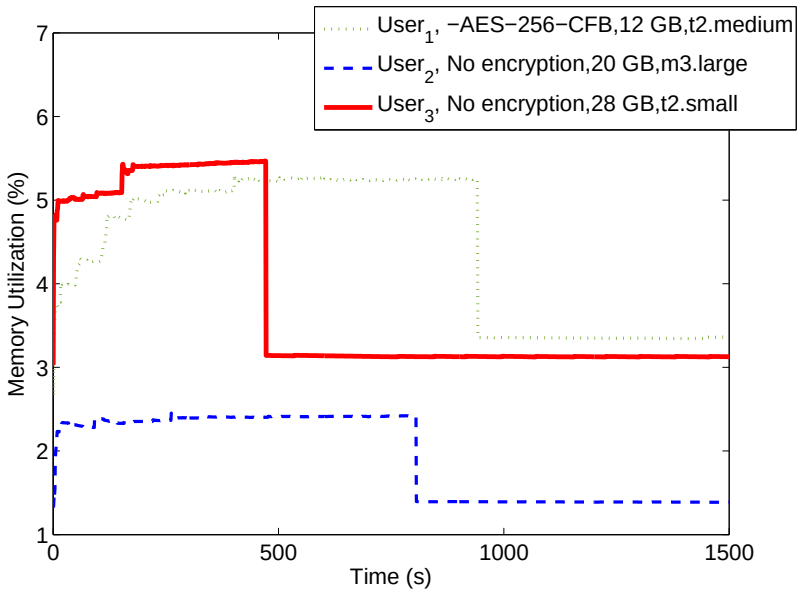

Figure 5.5: Memory utilization for all three users in Experiment # 1.

1103s. Fig. 5.12, shows the optimal performance for all users.

5.5 Conclusions

In this chapter, we have presented a formal mathematical approach to optimally distribute virtual resources in the cloud between a set of users. This novel technique uses the notion of tail probabilities and sample complexity to determine bounds of probability that a random variable, with a non-compact pdf, takes a value in the tail of the distribution far from the mean.

Some theoretical results that provide the sample complexity to solve an optimization algorithm given certain accuracy ϵ and confidence $1-\delta$, with $\epsilon, \delta \in (0,1)$ have been provided. Moreover, we introduced a heuristic algorithm for the parallelization of the optimization process given the sometimes prohibitive number of iterations that

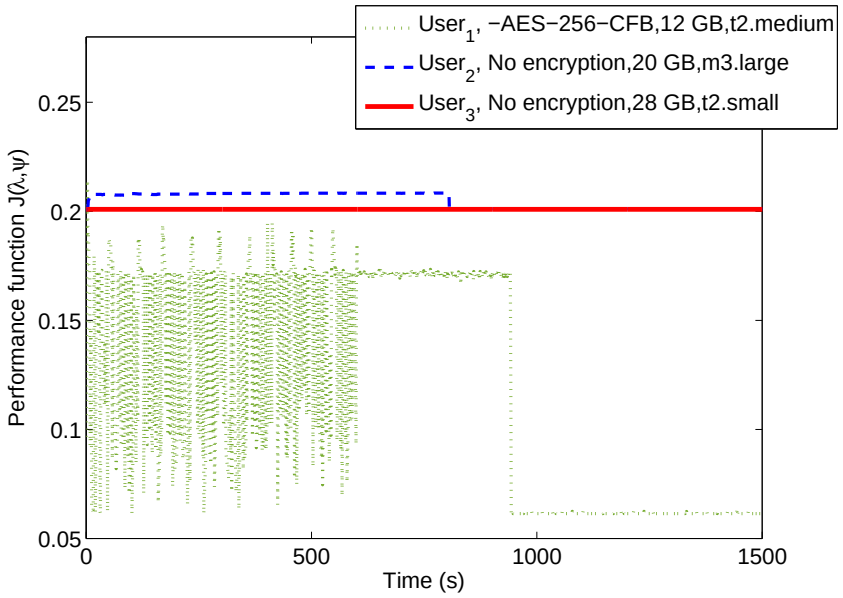

Figure 5.6: Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 1.

may be obtained from the sample complexity analysis.

Security has been introduced as part of the virtual resources to be optimized. This approach proposes a security metric consisting of an ordered classification of cryptographic algorithms based on their key-length, their capacity of hiding identification patterns, their immunity to cryptanalysis, the parallelization of the algorithm and their capacity of overcoming errors. Furthermore, this approach is compatible with the security metrics presented in [13, 14].

This approach has been implemented and tested in the AWS EC2 cloud, which is a commercial cloud widely used around the world. The results have been verified showing that this approach is able to optimize resources in open loop based on measured performance. Its implementation reflects not only its applicability but its compatibility with other closed-loop approaches.

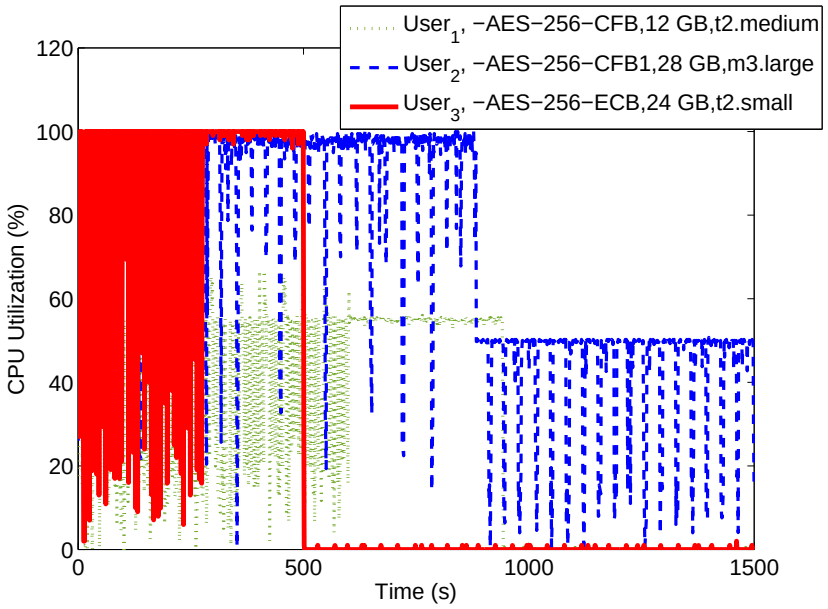

Figure 5.7: CPU utilization for all three users in Experiment # 2.

Furthermore, since every instance runs its own optimization problem, this approach is decentralized and therefore, scalable. The scalability of the algorithm is not affected by the number of users to be considered during the optimization because RAs are independent of the dimension of the performance function vector **J**.

Finally, this approach is compatible with closed-loop regulators previously proposed in the literature. Typical approaches involve model identification to adjust the parameters of the controller online while keeping the system stable. Our RA is able to provide optimal reference values or configurations to be regulated by the closed-loop controller.

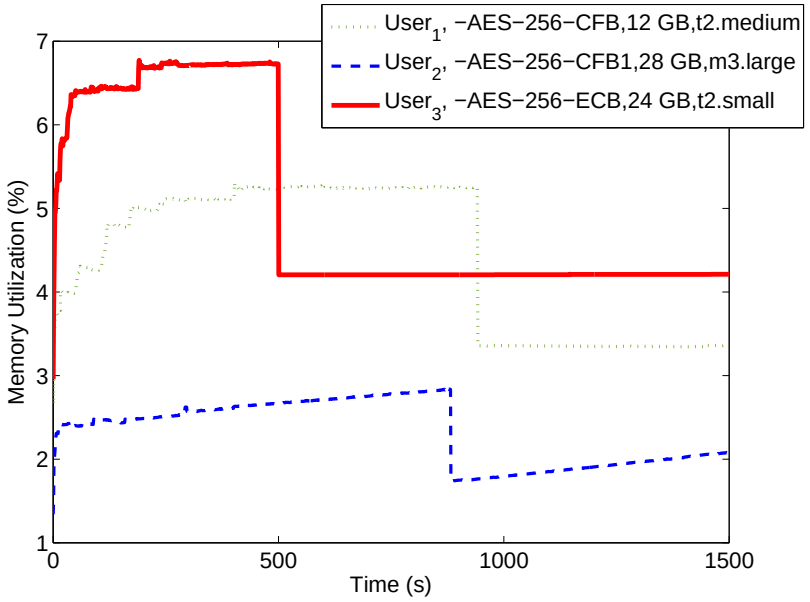

Figure 5.8: Memory utilization for all three users in Experiment # 2.

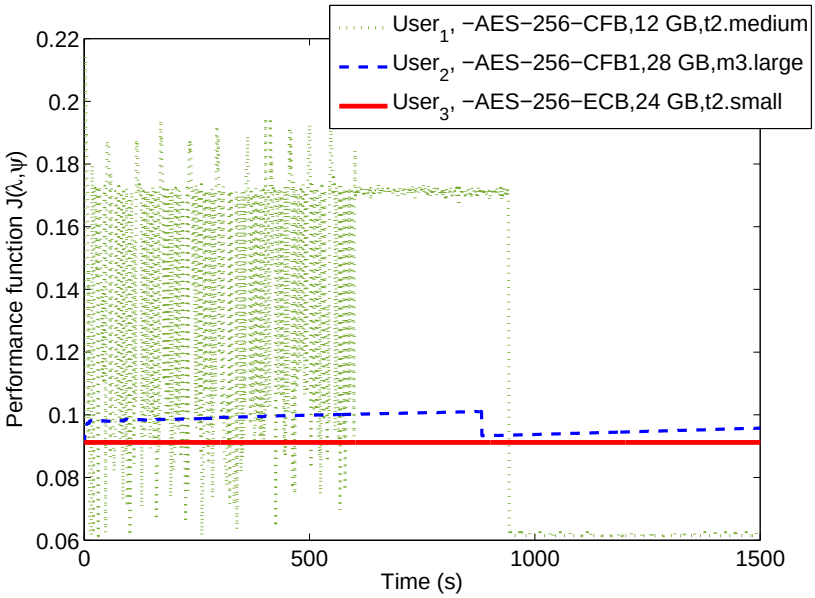

Figure 5.9: Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 2.

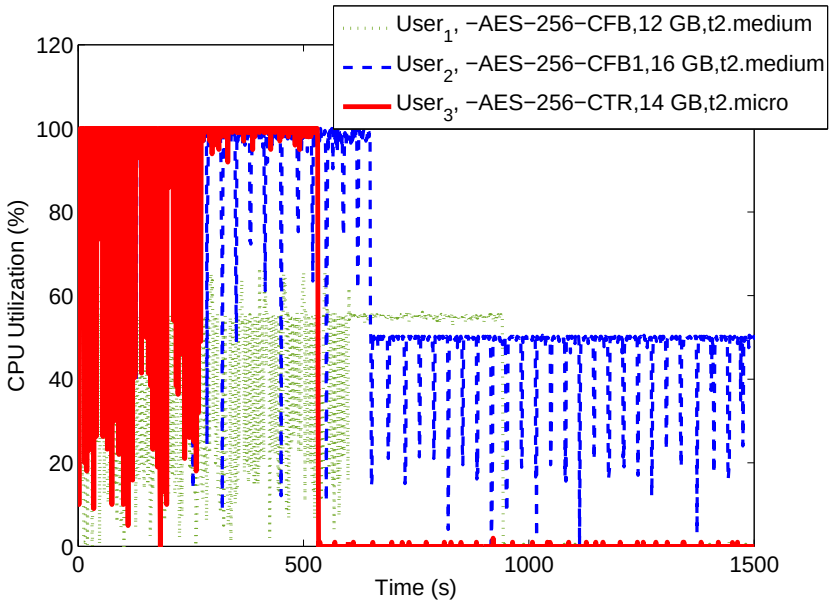

Figure 5.10: CPU utilization for all three users in Experiment # 3.

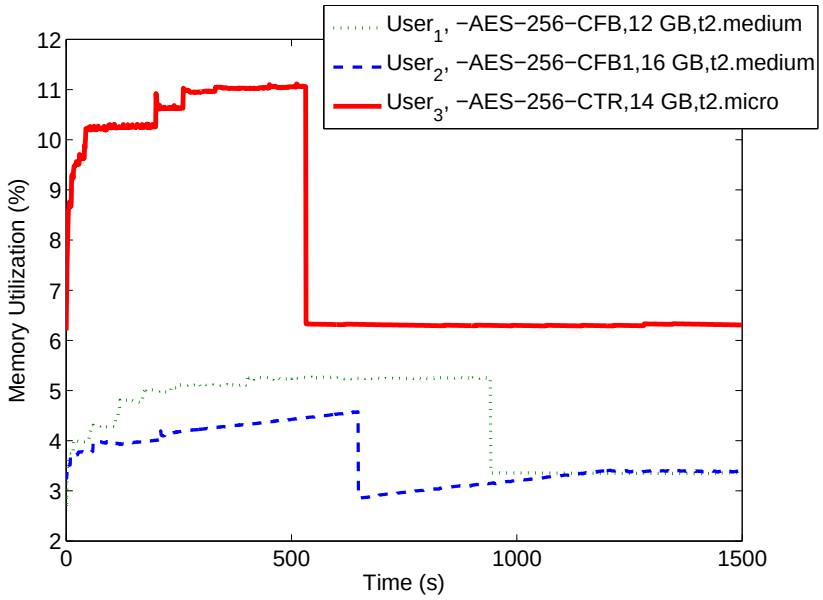

Figure 5.11: Memory utilization for all three users in Experiment # 3.

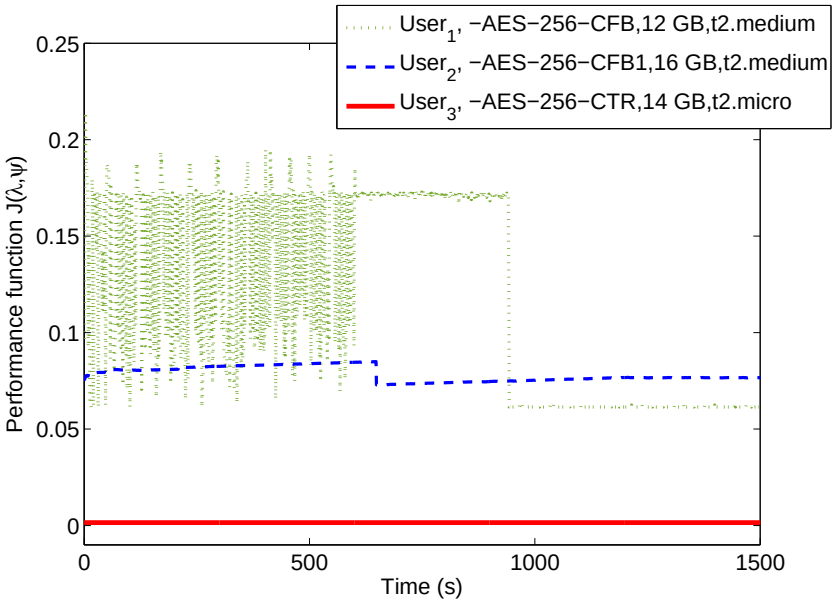

Figure 5.12: Optimal performance functions $J_k(\boldsymbol{\lambda}, \psi^*)$ for all three users in Experiment # 3.

Chapter 6

Concluding remarks, future work and recommendations

We have presented novel theoretical solutions to the problem of regulation and optimization of performance in computing systems. We have covered four case studies with different levels of complexity, namely, a) optimization and regulation of throughput and average system time in many-core processors, b) optimization and regulation of throughput in multitier servers, c) stability of a market-oriented cloud model, and d) optimization of CPU and memory utilization, hourly cost and security provisioning in IaaS in the cloud.

Our approach consists of two stages, namely, an open-loop optimizer and a closed-loop regulator. The optimizer is based on the implementation of an RA to calculate the optimal performance based on the specific requirements of the problem. The closed-loop regulator is based on IPA and keeps the performance measure in the vicinity of the optimal performance calculated by the open-loop optimizer.

6.1 Performance Optimization and Regulation for Many-Core Processors

6.1.1 Concluding Remarks

In the many-core processor problem, we used the out-of-order processor model previously proposed in [17] and calculated the IPA expressions to estimate the derivatives of the average system time and average waiting time. Furthermore, we provided the mathematical background and conditions for the implementation of the RA-based optimizer. We validated our theoretical approach by carrying out a simulation of four cores interacting with a slower peripheral using Matlab®. In order to reduce the wait states, and therefore the power consumption of the microprocessor we proposed a multi-objective function which minimized the throughput, the average system time and average waiting time of each core independently. After the optimal frequency and the optimal performance values were calculated, the IPA-based regulator kept the many-core processor regulated to linear combinations of the aforementioned performance metrics. The system remained regulated even when affected by statistical changes of its dynamics due to simulated variations of the running benchmark.

6.1.2 Future Work and Recommendations

- The simulation results that have validated the incorporation of statistical learning theory, as well as the addition of the average system time and average waiting time regulation were obtained using Matlab®. However, by carrying out simulations using highly detailed microprocessor simulators such as Zesto [78], recently incorporated to Manifold [79], the results will provide meaningful insights about the real behavior of the many-core processors due to variations of frequency and its repercussions over the power consumption of the system.

- Although complex in nature, the experimental implementation of a many-core processor architecture for validation is possible by using reconfigurable hardware, such as Field Programmable Gate Arrays (FPGAs) [80]. This implementation will provide really conclusive results about the applicability of our theoretical approach.

6.2 Performance Optimization and Regulation for Multitier Servers

6.2.1 Concluding Remarks

In the multi-tier case study, we used the queueing network model proposed in [52]. We present some additional mathematical assumptions and conditions to guarantee the unbiasedness of the IPA algorithm in order to estimate the derivative of the throughput of a multi-tier server. In this case, the open-loop optimizer calculated the optimal average service rate parameters of each server to guarantee that the IPA-based regulator was not affected by the low values of the service rates at the output of each tier. After the optimization process was carried out, the closed-loop regulator controlled the average service rate parameter of the front-end tier in order to regulate the throughput around a reference value assumed to be given *a priori*. Our algorithm was successfully simulated using Matlab®, showing that the system was able to optimize and regulate the performance of a three-tier server, even when the system was subject to statistical variations of its nominal parameters.

6.2.2 Future Work and Recommendations

- Several simplifications have been proposed in order to satisfy the mathematical conditions to guarantee unbiased IPA estimators. However, based on [52], the model may be enhanced to support contention control, load balancing and processor sharing [52]. With the implementation of queues with finite capacity, as well as load balancers and multiple servers per queue the queueing network is not guaranteed to satisfy the conditions for unbiased IPA estimates. However, even if the theoretical analysis is not viable, the current simulation results are limited by the stringent assumptions imposed over the model. Therefore, empirical simulation results involving the aforementioned modifications are worth to be carried out in order to explore further the applicability of our theoretical approach.

- Contention control is compatible with our proposed open-loop optimizer. The performance requirements can be easily adapted to the multi-objective function to guarantee the required performance measures based on the MPL. Therefore, a natural step forward once the queues with finite capacity have been incorporated to the model is the implementation of contention control in open loop using RAs.

6.3 Market-oriented Cloud Model with Time-varying Workload

6.3.1 Concluding Remarks

In the cloud computing case, we proceeded to enhance the passivity analysis proposed in [1]. Some sufficient conditions have been introduced to guarantee that the passivity

approach is valid in this context, and to correct the fact that an additional sufficient condition was required in order to guarantee that the system is asymptotically stable and ISS. The results have been verified by carrying out simulations in Matlab®.

6.3.2 Future Work and Recommendations

- Although this is a conceptual chapter, in principle, the approximated broker-server model may be implemented as an application running in the cloud. Therefore, we recommend to carry out the actual implementation of the dynamics of the brokers and servers described in [1] and [33] to verify their capabilities and services.

6.4 Optimal Performance and Security Provisioning in the Cloud

6.4.1 Concluding Remarks

In the problem of optimization of performance in the cloud, we only implemented the open-loop optimizer. Different to the previous cases, the RA of the optimizer is based on sample complexity of finite families. We provided some additional theoretical results that defined the steps of RA for this case study. For each client in the cloud, the performance metrics considered in the cost function were the CPU and memory utilization, the hourly cost of the instance, the volume size and the security level. We defined our security metric based on the performance of ten different cryptographic cyphers for data storage. This approach was validated through experiments carried out using AWS EC2. The system was able to satisfy the requirements given by the emulated users in a decentralized fashion. Although the possibility of implementing

an IPA-based closed-loop regulator is hindered by the coarse-grained granularity of the controllable parameters, this approach is compatible with the closed-loop approaches presented in [60, 61, 62].

6.4.2 Future Work and Recommendations

- The measure of security using cryptographic cyphers that was implemented in our case study may be modified by implementing the security level agreements proposed in [13, 14]. This approach incorporates additional cloud security levels that are expected to prove the viability of our optimization process under different security metrics.

- The experimental verification of our approach should be carried out with a larger number of clients in the cloud. The main purpose of these experiments is to prove that the scalability of the algorithm is not highly affected by the increase on the number of clients.

- The exploration of constrained optimization problems to improve the performance of the heuristic algorithm for execution time reduction may be carried out. Notice that by assuming that the optimization problem can be decomposed into subproblems, allows the potential implementation of a *dynamic programming* approach [81] in order to optimally reduce execution time.

Appendices

Event Time Derivatives for Three-Tier Servers **106**

Recalculation of $\hat{u}(k)$ for Market-oriented Cloud **108**

Heuristic Execution Time Reduction for Security and Performance Optimization in the Cloud **110**

Appendix A

Event Time Derivatives for Three-Tier Servers

The pseudo-code shown in Algorithm 5, illustrates the steps to carry out the event-time derivatives of the lifetimes associated to the three-tier problem with respect to μ_1. This three-tier problem is explained in Section 3.3.2.

Algorithm 5 Event time derivative for three-tier servers w.r.t μ_1.

Initial state $x_1 \leftarrow 0, x_2 \leftarrow 0, x_3 \leftarrow 0$, then event a is feasible and $\Delta_a \leftarrow 0$

Events $d_{10}, d_{12}, d_{21}, d_{23}$ and d_{32} are unfeasible,

then, $\Delta_{d_{10}} \leftarrow 0, \Delta_{d_{12}} \leftarrow 0, \Delta_{d_{21}} \leftarrow 0, \Delta_{d_{23}} \leftarrow 0, \Delta_{d_{32}} \leftarrow 0$

while Queueing network is in execution **do**
  if Event a is observed $\wedge$ $x_1 = 0$ **then**
    $\Delta_{d_{10}} \leftarrow \Delta_a + \frac{dV_{d_{10}}}{d\mu_1}$
    $\Delta_{d_{12}} \leftarrow \Delta_a + \frac{dV_{d_{12}}}{d\mu_1}$
  end if
  if Event d_{10} is observed **then**
    $\Delta_a \leftarrow \Delta_{d_{10}}$
    $\Delta_{d_{10}} \leftarrow \Delta_{d_{10}} + \frac{dV_{d_{10}}}{d\mu_1}$
    $\Delta_{d_{12}} \leftarrow \Delta_{d_{10}} + \frac{dV_{d_{12}}}{d\mu_1}$
  end if
  if Event d_{12} is observed **then**
    $\Delta_{d_{10}} \leftarrow \Delta_{d_{12}} + \frac{dV_{d_{10}}}{d\mu_1}$
    $\Delta_{d_{12}} \leftarrow \Delta_{d_{12}} + \frac{dV_{d_{12}}}{d\mu_1}$
    if $x_2 = 0$ **then**
      $\Delta_{d_{21}} \leftarrow \Delta_{d_{12}}$
      $\Delta_{d_{23}} \leftarrow \Delta_{d_{12}}$
    end if
  end if
  if Event d_{21} is observed **then**
    $\Delta_{d_{23}} \leftarrow \Delta_{d_{21}}$
    if $x_1 = 0$ **then**
      $\Delta_{d_{10}} \leftarrow \Delta_{d_{21}} + \frac{dV_{d_{10}}}{d\mu_1}$
      $\Delta_{d_{12}} \leftarrow \Delta_{d_{21}} + \frac{dV_{d_{12}}}{d\mu_1}$
    end if
  end if
  if Event d_{23} is observed **then**
    $\Delta_{d_{21}} \leftarrow \Delta_{d_{23}}$
    if $x_3 = 0$ **then**
      $\Delta_{d_{32}} \leftarrow \Delta_{d_{23}}$
    end if
  end if
  if Event d_{32} is observed $\wedge$ $x_2 = 0$ **then**
    $\Delta_{d_{21}} \leftarrow \Delta_{d_{32}}$
    $\Delta_{d_{23}} \leftarrow \Delta_{d_{32}}$
  end if
end while

Appendix B

Recalculation of $\hat{u}(k)$ for Market-oriented Cloud

In order to satisfy the sufficient conditions derived from the passivity approach in [1] some adjustments must be carried out in (4.5). By using the storage function (4.8) and assuming that projection in (4.3) is inactive the first difference of (4.8) gives,

$$\begin{aligned}
\Delta V_2 &\leq \xi^T(k+1)P\xi(k+1) - \xi^T(k)P\xi(k) \\
&= \xi^T(k)(A^TPA - P)\xi(k) + 2\xi^TA^TPB\hat{y}(k) \\
&\quad +\hat{y}(k)B^TPB\hat{y}(k) \\
&= 2\xi^TA^TPB\hat{y}(k) + \hat{y}(k)B^TPB\hat{y}(k) \\
&= 2\sigma b(k)\hat{y}(k) - \sigma s(k)\hat{y}(k) + \sigma\hat{y}^2(k) \\
&\leq 2\sigma b(k)\hat{y}(k) + 2\sigma s(k)\hat{y}(k) + \sigma\hat{y}^2(k) \\
&= \hat{u}_1(k)\hat{y}(k).
\end{aligned}$$

with,

$$\hat{u}_1(k) = 2\sigma b(k) + 2\sigma s(k) + \sigma\hat{y}(k).$$

Now, using the same storage function but assuming that the projection in (4.3) is active we obtain,

$$\begin{aligned}
\Delta V_2 &\leq \sigma(\sigma-1)(s(k)-b(k))^2 + 2\sigma b(k)\hat{y}(k) \\
&\quad +2\sigma s(k)\hat{y}(k) \\
&\leq 2\sigma b(k)\hat{y}(k) + 2\sigma s(k)\hat{y}(k) + \sigma\hat{y}^2(k) \\
&= \hat{u}_1(k)\hat{y}(k).
\end{aligned}$$

and the server system is still passive with the output (4.13).

Appendix C

Heuristic Execution Time Reduction for Security and Performance Optimization in the Cloud

The pseudo-code shown in Algorithm 6, shows the steps carried out by our heuristic algorithm to reduce the execution times of the experiments in the cloud. This algorithm is explained in Section 5.3.3.

Algorithm 6 Heuristics for Reduction of Execution Time

n_{par}, n_{samp} ▷ Number of parameters and samples
$n_{inst}, n_{vol}, n_{sec}$ ▷ Number of available resources
$inst[:] \leftarrow int(n_{inst} * rand(n_{par}))$ ▷ Ind. for instances
$vol[:] \leftarrow int(n_{vol} * rand(n_{par}))$ ▷ Ind. for volume size
$sec[:] \leftarrow int(n_{sec} * rand(n_{par}))$ ▷ Ind. for sec. levels
$ind[:][0] \leftarrow inst[:]$ ▷ Array of arrays for indices *ind*
$ind[:][1] \leftarrow vol[:]$
$ind[:][2] \leftarrow sec[:]$
$M[:][:]() \leftarrow 0$ ▷ Configuration matrix
▷ The entries are arrays indexed by ()
$M_a[:][:] \leftarrow$ 'avail' ▷ indicator matrix for available instances
$seq[]() \leftarrow []$ ▷ Output array of arrays with reduced cycles

for $i \leftarrow 0; i < n_{inst}; i^{++}$ **do** ▷ Building the configuration matrix
  for $j \leftarrow 0; j < n_{vol}; j^{++}$ **do**
    $M[i][j](:) \leftarrow ind\,[find\,(ind[:][0] = i$ **and** $ind[:][1] = j)][\,:]$ ▷ Locating seq. in M according to *inst* and *vol*
  end for
end for

$k \leftarrow 0$
while $M[:][:](:) \neq 0$ **do** ▷ Building the output $seq[]()$
  for $i \leftarrow 0; i < n_{inst}; i^{++}$ **do**
    for $j \leftarrow 0; j < n_{vol}; j^{++}$ **do**
      if $M[i][j]() \neq 0$ **and** $M_a[i][j] =$ 'avail' **then** ▷ Condition for available instance
        for $m \leftarrow 0; m < len(M[i][j]); m^{++}$ **do**
          $seq[k](l) \leftarrow M[i][j](m)$
          if $len(M[i][j]) > 1$ **then**
            k^{++}
            $l \leftarrow 0$
          end if
        end for
        $M[i][j] \leftarrow 0$ ▷ Discarding conf. from M
        $M_a[i][:] \leftarrow$ 'unavail' ▷ Blocking row
        $M_a[:][j] \leftarrow$ 'unavail' ▷ Blocking column
        l^{++}
      end if
    end for
  end for
  k^{++}
  $l \leftarrow 0$
  $M_{ava}[:][:] \leftarrow$ 'avail' ▷ Resetting M_{ava}
end while
return $seq[:](:)$

References

[1] M. D. Lemmon, "Towards a passivity framework for power control and response time management in cloud computing," in *Proceedings of the International Workshop on Feedback Computing*, San Jose, CA, September 2012.

[2] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu, "Introduction to control theory and its application to computing systems," in *Performance Modeling and Engineering*. Springer, 2008, pp. 185–215.

[3] C. Poussot-Vassal, M. Tanelli, and M. Lovera, "Linear parametrically varying mpc for combined quality of service and energy management in web service systems," in *Proceedings of the American Control Conference (ACC-2010)*, Baltimore, MD, June 2010, pp. 3106–3111.

[4] Z. Wang, N. Tolia, and C. Bash, "Opportunities and challenges to unify workload, power, and cooling management in data centers," in *Proceedings of the ACM International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (Febid-2010)*, Paris, April 2010, pp. 1–6.

[5] D. Hilley, "Cloud computing: A taxonomy of platform and infrastructure-level offerings," Georgia Institute of Technology, Tech. Rep., April 2009.

[6] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the ACM European Society in Systems Conference 2010*, Paris, France, April 2010, pp. 237–250.

[7] B. D. Chung, H. Jeon, and K.-K. Seo, "A framework of cloud service quality evaluation system - focusing on security quality evaluation," *International Journal of Software Engineering and its Applications*, vol. 8, no. 4, pp. 41–46, May 2014.

[8] D. Zissis and D. Lekkas, "Addressing cloud computing security issues," *Future Generation Computer Systems*, vol. 28, no. 3, pp. 583–592, 2012.

[9] B. R. Kandukuri, V. R. Paturi, and A. Rakshit, "Cloud security issues," in *IEEE International Conference on Services Computing (SCC-09)*, Bangalore, India, September 2009, pp. 517–520.

[10] S. Pearson and A. Benameur, "Privacy, security and trust issues arising from cloud computing," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom-2010)*, Indianapolis, IN, November 2010, pp. 693–702.

[11] Y. Diao, J. L. Hellerstein, S. Parekh, H. Shaikh, and M. Surendra, "Controlling quality of service in multi-tier web applications," in *IEEE International Conference on Distributed Computing Systems (ICDCS-2006)*, 2006, pp. 25–32.

[12] M. Dekker and G. Hogben, "Survey and analysis of security parameters in cloud slas across the european public sector," 2011. [Online]. Available: http://www.enisa.europa.eu/activities/Resilience-and-CIIP/cloud-computing/survey-and-analysisof-security-parameters-in-cloud-slas-across-the-european-public-sector

[13] J. Luna, H. Ghani, T. Vateva, and N. Suri, "Quantitative assessment of cloud security level agreements: A case study," Rome, Italy, July 2012.

[14] J. Luna, R. Langenberg, and N. Suri, "Benchmarking cloud security level agreements using quantitative policy trees," in *Proceedings of the ACM Workshop on Cloud computing security*, Raleigh, NC, October 2012, pp. 103–112.

[15] J. M. Luna and C. T. Abdallah, "Control in computing systems: Part i," in *IEEE International Symposium on Computer-Aided Control System Design (CACSD-2011)*, Denver, CO, September 2011, pp. 25–31.

[16] ——, "Control in computing systems: Part ii," in *IEEE International Symposium on Computer-Aided Control System Design (CACSD-2011)*, Denver, CO, September 2011, pp. 32–36.

[17] N. Almoosa, W. Song, Y. S. Yalamanchili, and Y. Wardi, "Throughput regulation in multicore processors via ipa," in *Proceedings of the American Control Conference (ACC-2012)*, Montreal, June 2012, pp. 7267–7272.

[18] N. Almoosa, W. Song, Y. Wardi, and S. Yalamanchili, "A power capping controller for multicore processors," in *Proceedings of the American Control Conference (ACC-2012)*, Montreal, 2012, pp. 4709–4714.

[19] Y. Zhang, Y. Wang, and X. Wang, "Capping the electricity cost of cloud-scale data centers with impacts on power markets," in *Proceedings of the International*

Symposium on High Performance Distributed Computing, San Jose, CA, June 2011, pp. 271–272.

[20] Z. Wu, C. Giles, and J. Wang, "Classified power capping by network distribution trees for green computing," *Cluster computing*, vol. 16, no. 1, pp. 17–26, 2013.

[21] H. Yuan, C.-C. Kuo, and I. Ahmad, "Energy efficiency in data centers and cloud-based multimedia services: An overview and future directions," in *International Green Computing Conference*, Chicago, IL, August 2010, pp. 375–382.

[22] Z. Abbasi, T. Mukherjee, G. Varsamopoulos, and S. K. S. gupta, "Dynamic hosting management of web based applications over clouds," in *Proceedings of the International Conference on High Performance Computing*, Tempe, AZ, December 2011, pp. 1–10.

[23] J. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[24] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, "Topology-aware resource allocation for data-intensive workloads," in *Proceedings of the ACM Asia-Pacific Workshop on Systems (ApSys-2010)*, New Delhi, India, August 2010, pp. 1–6.

[25] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu, "Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures," in *IEEE International Conference on Distributed Computing Systems (ICDCS-2010)*, Genoa, June 2010, pp. 62–73.

[26] S. Haykin, *Adaptive Filter Theory*. Prentice Hall, 2002.

[27] C. G. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C. G. Payaniotou, "Perturbation analysis for on-line control and optimization of stochastic fluid models," *IEEE Transactions on Automatic Control*, vol. AC-47, no. 8, pp. 1234–1248, 2002.

[28] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. New York: Springer, 2008.

[29] Y. Wardi and C. Seatzu, "Infinitesimal perturbation analysis of stochastic hybrid systems: Application to congestion management in traffic-light intersections," *arXiv preprint arXiv:1407.7200*, 2014.

[30] C. Seatzu and Y. Wardi, "Performance regulation via integral control in a class of stochastic discrete event dynamic systems," in *Proceedings of the IFAC/IEEE Workshop on Discrete Event Systems*, Cachan, France, May 2014, pp. 259–264.

[31] R. Tempo, G. Calafiore, and F. Dabbene, *Randomized Algorithms for Analysis and Control of Uncertain Systems, with Applications*, 2nd ed. London: Springer-Verlag, 2013.

[32] M. Vidyasagar, "Statistical learning theory and randomized algorithms for control," *IEEE Control Systems Magazine*, vol. 18, pp. 69–85, 1998.

[33] J. M. Luna, C. T. Abdallah, and G. L. Heileman, "On the stability of a market-oriented cloud computing model with time-varying workloads," in *International Workshop on Feedback Computing*, San Jose, CA, June 2013.

[34] ——, "Probabilistic resource optimization and security provisioning in the cloud," *IEEE Transactions on Cloud Computing*, 2014, submitted.

[35] G. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[36] R. Dennard, F. H. Gaensslen, V. L. R. Hwa-Nien Yu, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Solid-State Circuits Society Newsletter*, vol. 12, no. 1, pp. 38–50, 2007.

[37] M. Pedram and S. Nazarian, "Theral modeling, analysis and management in vlsi circuits: Principles and methods," *Proceedings of the IEEE*, vol. 94, no. 8, pp. 1487–1501, August 2006.

[38] P. Bose, *Encyclopedia of Parallel Computing*. Springer US, 2011, ch. Power Wall, pp. 1593–1608. [Online]. Available: http://link.springer.com/referenceworkentry/10.1007%2F978-0-387-09766-4_499

[39] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *IEEE Symposium on Low Power Electronics. Digest of Technical Papers*, San Diego, October 1994, pp. 8–11.

[40] T. D. Burd and R. W. Brodersen, "Energy efficient cmos microprocessor design," in *Proceedings of the International Conference on System Sciences*, Wailea, HI, January 1995, pp. 288–297.

[41] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark, "Formal online methods for voltage/frequency control in multiple clock domain microprocessors," *ACM SIGARCH Computer Architecture News*, vol. 32, no. 5, pp. 248–259, 2004.

[42] P. Juang, Q. Wu, L.-S. Peh, M. Martonosi, and D. W. Clark, "Coordinated, distributed, formal energy management of chip multiprocessors," in *Proceedings of*

the International Symposium on Low Power Electronics and Design (ISLPED-2005), New York, 2005, pp. 127–130.

[43] Y. Zhu and F. Mueller, "Exploiting synchronous and asynchronous dvs for feedback edf scheduling on an embedded platform," *ACM Transactions on Embedded Computing Systems*, vol. 7, no. 1, pp. 3:1–3:26, December 2007.

[44] J. Suh and M. Dubois, "Dynamic mips rate stabilization in out-of-order processors," in *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, 2009, pp. 46–56.

[45] Y.-C. Ho and X.-R. Cao, *Perturbation Analysis of Discrete Event Dynamic Systems*. Boston: Kluwer Academic Publishers, 1991.

[46] C. G. Cassandras, *Stochastic Hybrid Systems: Recent Developments and Research Trends*. New York: CRC Press, 2006, ch. Stochastic Flow Systems: Modeling and Sensitivity Analysis, pp. 139–167.

[47] T. Alamo, R. Tempo, A. Luque, and D. R. Ramírez, "The sample complexity of randomized methods for analysis and design of uncertain systems," *arXiv preprint arXiv:1304.0678*, 2013.

[48] S. Ball, *Embedded Microprocessor Systems: Real World Design*, 3rd ed. Newton, MA, USA: Butterworth-Heinemann, 2002.

[49] E. Le Sueur and G. Heiser, "Dynamic voltage and frequency scaling: The laws of diminishing returns," in *Proceedings of the Workshop on Power Aware Computing and Systems (HotPower-2010)*, Vancouver, October 2010, pp. 1–8.

[50] L. Malrait, "Qos-oriented control of server systems," in *Proceedings of the International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks*, Paris, April 2010, pp. 16–21.

[51] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 560–561, June 2005.

[52] B. Urgaonkar, G. Pacifi, P. Shenoy, M. Spreitzer, and A. Tantawi, "Analytic modeling of multitier internet applications," *ACM Transactions on the Web*, vol. 1, no. 1, May 2007.

[53] S. Parekh, N. Gandhi, J. Hellerstein, D. Tilbury, T. Jayram, and J. Bigus, "Using control theory to achieve service level objectives in performance management," *Real-Time Systems*, vol. 23, no. 1-2, pp. 127–141, 2002.

[54] S. Ranjan, J. Rolia, H. Fu, and E. Knightly, "Qos-driven server migration for internet data centers," in *IEEE International Workshop on Quality of Service*, 2002, pp. 3–12.

[55] D. Villela, P. Pradhan, and D. Rubenstein, "Provisioning servers in the application tier for e-commerce systems," *ACM Transactions on Internet Technology*, vol. 7, no. 1, p. 7, 2007.

[56] A. Robertsson, B. Wittenmark, M. Kihl, and M. Andersson, "Admission control for web server systems-design and experimental evaluation," in *IEEE Conference on Decision and Control (CDC-2004)*, vol. 1, 2004, pp. 531–536.

[57] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queuing networks," *Journal of the ACM*, vol. 27, no. 2, pp. 313–322, 1980.

[58] P. Glasserman, *Gradient estimation via perturbation analysis*. Springer, 1991.

[59] H. Stark and J. W. Woods, *Probability and random processes with applications to signal processing*. Upper Saddle River, N.J. Prentice Hall, 2002.

[60] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: a generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the international conference on Autonomic computing*, Washington DC, June 2010, pp. 79–88.

[61] M. Maggio, H. Hoffmann, M. D. Santambrogio, A. Agarwal, and A. Leva, "Controlling software applications via resource allocation within the heartbeats framework," in *Proceedings of the IEEE Conference on Decision and Control (CDC-2010)*, Atlanta, GA, December 2010, pp. 3736–3741.

[62] H. C. Lim, S. Babu, J. S. Chase, and S. S. Parekh, "Automated control in cloud computing: Challenges and opportunities," in *Proceedings of the Workshop on Automatic Control for Datacenters and Clouds*, New York, June 2009, pp. 13–18.

[63] J. V. Vliet and F. Paganelli, *Programming Amazon EC2*. Oŕeilly Media Inc., 2011.

[64] J. Yao, X. Liu, X. Chen, X. Wang, and J. Li, "Online decentralized adaptive optimal controller design of cpu utilization for distributed real-time embedded systems," in *Proceedings of the American Control Conference (ACC-2010)*, Baltimore, MD, June 2010, pp. 283–288.

[65] R. Kulhavý, "Restricted exponential forgetting in real-time identification," *Automatica*, vol. 25, no. 5, pp. 589–600, 1987.

[66] S. Subashini and V. Kavitha, "A survey on security issues in service delivery models of cloud computing." *Journal Network and Computer Applications*, vol. 34, no. 1, pp. 1–11, 2011.

[67] D. Chen and H. Zhao, "Data security and privacy protection issues in cloud computing," in *Proceedings of the International Conference on Computer Science and Electronics Engineering - (ICCSEE-2012)*, Washington, DC, USA, 2012, pp. 647–651.

[68] R. Tempo, E.-W. Bai, and F. Dabbene, "Probabilistic robustness analysis: Explicit bounds for the minimum number of samples," in *Proceedings of the IEEE Conference on Decision and Control (CDC-1996)*, vol. 3, 1996, pp. 3424–3428.

[69] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta, "Modeling virtualized applications using machine learning techniques," in *Proceedings of the ACM SIGPLAN/SIGOPS Conference on Virtual Execution Environments*, London, England, UK, 2012, pp. 3–14.

[70] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini, "Dejavu: Accelerating resource allocation in virtualized environments," *SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 423–436, Mar. 2012.

[71] "System monitor," 2013. [Online]. Available: https://apps.ubuntu.com/cat/applications/quantal/gnome-system-monitor/

[72] "View cpu utilization and other performance information," 2014. [Online]. Available: http://windows.microsoft.com/en-us/windows/view-cpu-utilization-performance-information\#1TC=windows-7

[73] "Amazon ec2 instances," 2014. [Online]. Available: http://aws.amazon.com/ec2/instance-types/

[74] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks: Java benchmarking development and analysis," in *Proceedings of the annual ACM SIGPLAN conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA-2006)*, Portland, OR, USA, October 2006, pp. 169–190.

[75] J. Viega, M. Messier, and P. Chandra, *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, Inc., 2002.

[76] M. Lutz, *Programming python*. O'Reilly Media, Inc., 2010.

[77] J. Murty, *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. " O'Reilly Media, Inc.", 2008.

[78] G. Loh, S. Subramanian, and Y. Xie, "A cycle-level simulator for highly detailed microarchitecture exploration," in *Proceedings of the International Conference on the Performance Analysis of Software Systems (ISPASS-2009)*, Boston, April 2009, pp. 53–64.

[79] J. Wang, "Manifold: A parallel simulation framework for multicore systems," Ph.D. dissertation, Georgia Institute of Technology, 2011.

[80] R. J. Francis, J. Rose, and Z. G. Vranesic, *Field-programmable gate arrays*. Springer, 1992, vol. 180.

[81] D. Bertsekas, *Dynamic programming and optimal control*, 4th ed. Belmont, MA: Athena Scientific, 2007.