FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Energy-efficient and SLA-based Management of IaaS Cloud Data Centers

Altino Manuel Silva Sampaio

Programa Doutoral em Engenharia Informática

Supervisor: Jorge Manuel Gomes Barbosa

June 23, 2015

Energy-efficient and SLA-based Management of IaaS Cloud Data Centers

Altino Manuel Silva Sampaio

Programa Doutoral em Engenharia Informática

June 23, 2015

Resumo

A computação em nuvem tem vindo a ser progressivamente adoptada em diversos cenários, oferecendo acesso a pedido a um ambiente de recursos distribuídos de larga escala, de forma flexível e altamente escalável, orientada ao acordo de nível de serviço. A virtualização é a tecnologia base dos sistemas de computação em nuvem, dotando-os de serviços flexíveis e escaláveis. À medida que estes sistemas distribuídos se tornam mais comuns, as empresas e fornecedores destes sistemas constrõem maiores centros de dados para lidar com a crescente procura por recursos de computação. No entanto, a quantidade de energia elétrica consumida por estes centros de dados aumenta com o poder computacional instalado. Na mesma linha, à medida que os sistemas de computação crescem em tamanho e em complexidade, eventos de falha tornam-se norma em vez de exceção, aumentando ainda mais o desperdício energético e afetando a Qualidade de Serviço do sistema percecionado pelo utilizador final. Além disso, as tecnologias de virtualização atuais não fornecem isolamento de desempenho, o que significa que duas aplicações executando em máquinas virtuais independentes e que partilham o mesmo servidor podem interferir na execução uma da outra, e consequentemente violando as restrições de Qualidade de Serviço.

Esta tese apresenta dois mecanismos aperfeiçoados com o duplo objetivo de reduzir os custos de energia elétrica, e respeitando o acordo de nível de serviço estipulado entre os operadores e seus utilizadores. O primeiro objetivo é alcançado aliando um mecanismo de energia que deteta e mitiga ineficiências energéticas, com ferramentas de virtualização para implementar clusters virtuais com tolerância pro-ativa a falhas e eficiência energética. As ineficiências energéticas são reduzidas através da consolidação dinâmica das máquinas virtuais e por ativação/desativação de servidores de acordo com a solicitação de recursos. A consolidação de máquinas virtuais é implementada através da eslasticidade vertical e horizontal de recursos. O segundo objetivo é atingido por articulação entre um mecanismo de estimativa de desempenho, que deteta desvios entre a Qualidade de Serviço requerida e efetivamente atribuída a uma aplicação, e ferramentas da virtualização para adaptar o mapeamento máquinas virtuais – servidores. Dois tipos de cargas são consideradas, nomeadamente cargas de trabalho intensivas em termos de CPU, e cargas de trabalho intensivas em termos de entrada/saída de rede, ambas com diferentes restrições de Qualidade de Serviço. A análise ao desempenho dos mecanismos propostos é feita via simulação e testes reais em ambiente de laboratório. As caraterísticas das cargas, falhas, e desempenho usadas nos ensaios são baseadas nos atributos e propriedades descritas em estudos e análises atuais sobre centros de dados de larga escala. No caso do primeiro objetivo, os resultados indicam que a estratégia proposta optimiza a relação trabalho por Joule em aproximadamente 12.9% e a eficiência do trabalho é melhorada em quase 15.9%, quando comparadas com algoritmos atuais. Para o segundo objetivo, os resultados mostram que o mecanismo de aplicação de desempenho proposto é capaz de cumprir com o acordo de nível de serviço contratado em ambientes do mundo real, ao mesmo tempo que reduz o custo com a energia até 21%.

Abstract

Cloud computing is progressively being adopted in different scenarios by offering on-demand, flexible, and high-scalability access to large-scale distributed resources, with Service Level Agreements-driven management. Virtualization is the basic technology of cloud computing, rendering flexible and scalable system services to cloud systems. As these distributed systems become more widespread, companies and resource providers are building large warehouse-sized data centers to cope with increasing demand for computing resources. However, the amount of electrical energy consumed by data centers increases with the amount of computing power instaled. In the same line, as compute systems grow in size and in complexity, failure events become norm instead of exception, increasing the energy waste even more and affecting the Quality-of-Service of the system perceived by end-users. Moreover, current virtualization technologies do not provide performance isolation, meaning that two applications running in independent virtual machines can interfere in the execution of each other when they share the same physical server, hence violating the Quality-of-Service constraints.

This thesis presents two improved mechanisms with the twofold objective of saving electrical costs and respecting the Service Level Agreements stipulated with users. The first objective is achieved by allying an energy optimizing mechanism to detect and mitigate energy inefficiencies, and virtualization tools to provide proactive fault-tolerance and energy efficiency to virtual clusters. Energy inefficiencies are reduced by dynamically consolidating virtual machines and switching off and on physical nodes according to resource demand. Consolidation is implemented based on vertical and horizontal elasticity of resources. The second objective is achieved by articulating a performance estimator mechanism to detect deviation from application Quality-of-Service requirements, and virtualization tools to adapt the map of virtual machines to servers. Two types of workloads are considered, namely CPU- and network-bound workloads, with different Quality-of-Service constraints. The analysis of the performance of the proposed mechanisms is done via simulation and experiments in real cloud testbed. The workloads, failures, and performance characteristics used in tests are coherent with the attributes outlined in state-of-the-art studies over large-scale data centers. In the case of the first objective, the results indicate that the proposed strategy improves the work per Joule ratio by approximately 12.9% and the working efficiency by almost 15.9% compared with other state-of-the-art algorithms. For the second objective, the results show that the proposed performance enforcing mechanism is able to fulfil contracted SLAs of real-world environments, while reducing energy costs up to 21%.

Acknowledgements

Thomas A. Edison once said: *"Our greatest weakness lies in giving up. The most certain way to succeed is always to try just one more time."*. Pursuing a PhD is an exhausting, and emotional struggling. A long way to go, and yet the path for creative freedom. I did not give up trying, and I learnt a lot. I am truly happy that I have had the opportunity to complete it. It would not have been possible without all those people who helped me along the way. I am greatly thankful to my supervisor, Professor Jorge G. Barbosa, who was always available to help me and guide me along the way, and provided with invaluable advices throughout my PhD candidature.

I wish to acknowledge IBM Portugal Center for Advanced Studies for providing access to a high-performance IBM Cluster, where the real platform experiments were performed, the Faculty of Engineering of University of Porto and LIACC Laboratory for travel support, and Instituto Politécnico do Porto and CIICESI research center, for flexibility of working schedule and for the travel support which enabled me to pursue my doctoral studies and attend international conferences.

I would like to give thanks to my parents, my brother and sister for their constant support, encouraging words, love, and endless help. Finally, I would like to extend my heartfelt thanks to my wife Sasha, for her love, patience, and constant inspiration, and for all the joy and happiness she brought to my life.

Altino Manuel Silva Sampaio

"Attitude is a little thing that makes a big difference."

Winston Churchill

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CLOUD computing has proven to be a major commercial success over recent years by enabling on-demand provisioning of elastic computing resources on a pay-as-you-go basis. It refers to both the applications delivered as services over the Internet and the hardware and systems software in the data centers that sustain such services. Cloud computing's aim is to make a better use of distributed resources, which the massive computers constitute, while offering dynamic flexible infrastructures and Quality-of-Service (QoS) guaranteed services. From a hardware point of view, it gives users the illusion of infinite computing resources available on demand. Cloud infrastructures allow organizations to outsource their computational needs so as to reduce management overhead and to minimize investment costs in private computing infrastructures, or to extend existing limited private infrastructures to improve the resource management and provisioning processes. Even more importantly, cloud systems allow providers to offer further types of applications to a wide market whilst minimizing the entry costs.

Virtualization is considered the base of cloud computing. The intrinsic nature of system virtualization technologies renders flexible and scalable system services to cloud systems, creating a powerful computing environment where virtualized resources can be dynamically allocated, expanded, shrunk or moved as demand varies. Based on virtualization, cloud computing addresses most important inherent features such as scalability, interoperability, failover, load balancing, etc. Today's virtualization technologies enable virtualization of many factors, such as Information Technology (IT) resources, hardware, software, operating systems (OSs) and storage, whereas environment is decoupled from physical platform.

In spite of the advantages brought by the cloud computing emerging business model, nowadays IT industry is adopting cloud to offer users high-available, reliable, scalable and inexpensive dynamic computing environments. However, proliferation of cloud computing and consequently the establishment of large-scale data centers all over the world containing massive number of interconnected compute nodes brought new challenges to the management of these computing environments: failures and energy consumption. Node failures are a characteristic of very large-scale distributed systems, that can have thousands of nodes running a variety of different jobs. A study conducted by Los Alamos National Laboratory (LANL) [Phi05] estimated that a node's

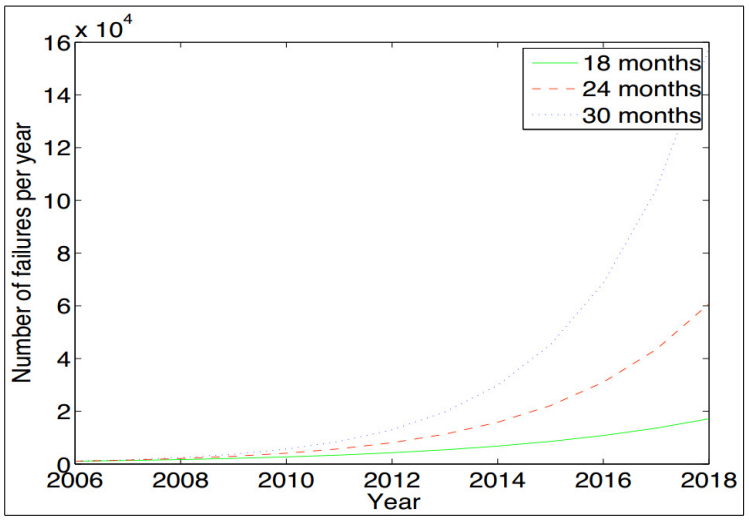


Figure 1.1: The expected growth in failure rate [SG07].

mean time between failures (MTBF) is 1.25 h in a petaflop system. In Figure 1.1 is shown the expected increase in failure rate, with numbers of cores doubling every 18, 24 and 30 months. Today's available information suggests that failure occurrences will become far more common in the coming decade, severely impacting the system performance and operation costs.

On the other hand, as cloud computing becomes more widespread, the amount of electrical energy consumed by data centers increases, resulting in high operational costs and emission of carbon dioxide (CO_2) to the environment. This approach creates great pressure for cloud providers that need to reduce operational costs through improved energy utilization. As shown in Figure 1.2, the electricity used by data centers worldwide increased by about 56% from 2005 to 2010, and in 2010 was accounted for between 1.1% and 1.5% of total electricity use. Moreover, Koomey [Koo11] estimates that energy consumption in data centers will continue to grow in the future. Energy-efficient resource management solutions must be developed to help reducing the energy consumption.

Failure occurrences and power consumption have become a critical concern in designing modern cloud systems, because the success of petascale computing will depend on the ability to provide dependability at scale, at acceptable operational costs. To address the problem of failure occurrences, adopting failure-aware resource management is crucial. Failure prediction technique can be used to proactive and autonomically construct adaptive computing infrastructures, that self-adjusts their configuration dynamically at runtime according to the components' availability. In turn, eliminating inefficiencies and waste in the way, resources are utilized to serve applications, workloads help improving energy efficiency. The Open Compute Project has reported recently that Facebook's Prineville, Oregon, data center presents a Power Usage Effectiveness (PUE) of 1.08 (PUE concept is defined in Section 3.2.2), which means that about 92% of energy from the data center is consumed directly by the computing resources [Pro11]. Also Forest City data center is reported having a PUE of 1.09. These numbers suggest that source of energy waste occurs due to the inefficient usage of computing resources. Garraghan et al. [GTX13] have analysed

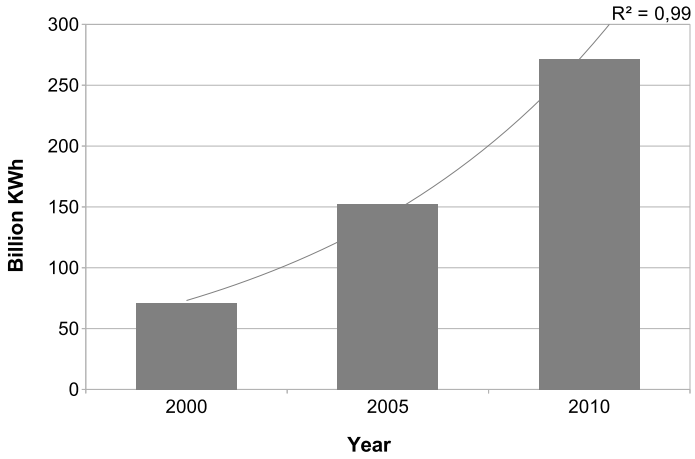


Figure 1.2: Total electricity used by data centers worldwide [Koo11].

the resource utilization in Google cloud data center and have concluded that resource utilization remains between 40 – 60%. Additionally, the same authors concluded that the amount of resource utilization wasted due to task termination and consequent preceding work being lost varies between 4.53–14.22%. Considering that an idle server consumes from 50% to 70% of its total power [BGDG+10], cloud users' applications can be consolidated and servers underutilized can go into sleep mode in order to reduce the energy consumption.

Guarantee the expected QoS for applications, and reduce the energy consumption via consolidation at the same time, is challenging because of performance deviations. Technological limitations related to virtualization, and diverse scheduling policies are the root of the problem. Despite the enormous progress in virtualization capabilities in the last years, the technology is still unable to handle performance isolation among virtual machine (VM) instances sharing the same physical host. Several studies (e.g., [CKK11, MLPS10, SC09, ZT12]) have studied this phenomenon, and concluded that, in the case of deadline-driven applications, the performance interference among co-hosted workloads can cause a 120% slowdown in the execution of applications. Another problem regarding QoS deviations is related to the type of workload a VM is running, which makes previous scheduling conditions outdated. In the case of web applications, in which workloads demand are bursty, outdated schedules can result in situations of scarcity of resources, with consequent penalty in QoS of applications perceived by end-users. On the other hand, over-provisioning of resources to applications is not a solution in the sense that it will result in waste of energy. This problem is even more serious since current cloud data centers either do not offer any performance guarantee or prefer static VM allocation over dynamic. For example, Amazon EC2 offers only guarantees on availability of resources, not on performance of applications running within VMs [GTGB14]. In this regard, it is imperative to consider these limitations in the scheduling of applications in the cloud.

1.1 Motivation and Scope

In a first stage, the scope of this thesis is creating and managing energy-efficient and fault-tolerant Infrastructure-as-a-Service (IaaS) cloud data centers under QoS constraints. In typical cloud computing systems, IaaS providers directly partition the physical resources into VMs, over which applications are deployed. Then, VMs are managed according to established policies. Consolidation of VM instances has been shown to be effective in improving utilization of resources [MMP13, XF10, XF11, ZZA10, RJQ+10, BAB12], because power consumption of the computing resources is linearly related to the application workload and consequently to the CPU utilization either [XF10]. One capability provided by virtualization is VM migration, an extremely powerful management tool allowing administrators to move a VM instance to another server in a seamless way. Also, several virtualization technologies deploy a cap mechanism, that specify a hard limit to implement an upper bound of resource consumption, to control the amount of CPU resource a VM instance can use. This thesis uses both migration and cap mechanisms to implement vertical and horizontal dynamic consolidation [SHRE13] of CPU-bound tasks executing within VMs, to improve the utilization of resources and reduce energy consumption.

An essential property for effective consolidation is to be able to efficiently utilize system resources for high-availability computing and energy efficiency optimization, because both factors have impact in energy consumption and QoS of the system perceived by end-users. Virtualization technology is increasingly being utilized to guarantee availability properties, in alternative to traditional expensive hardware solutions. Despite the existence of different fault tolerance schemes, there is a shift onto failure prediction tools [Fu10] in recent research on systems dependability to implement proactive fault tolerance, in response to excessive overhead traditional schemes imply [Phi05]. By continuously analysing performance data in a system, a resource manager can be applied to help determine possible occurrences of fatal events in the near future. In this way, proactive fault tolerance can be promoted by migrating VMs from health-deteriorating node to a healthy one for failure resilience and improving system availability.

However, while VM consolidation contributes to improved energy efficiency in a data center, current virtualization technologies do not provide performance isolation among co-hosted VM instances. To this end, in a second stage this thesis aims at creating and managing energy-efficient and interference-aware IaaS cloud data centers. Deviation in performance can occur due to performance interference among co-hosted VMs executing different workload types with diverse QoS constraints, that contend for hardware resources. CPU-bound contending for on-chip [HL13] resources and scheduling policies associated with the time characteristics of web workloads [GTGB14], for example, are the cause of the problem. Notable research efforts in this field delivered some performance interference meters [DFB+12] to measure the performance deviation with some error due to contention in on-chip resources. This thesis combines these state-of-the-art performance interference meters, with VM capabilities (i.e., migration and cap limits), to build interference- and power-aware IaaS cloud environments.

In order to make decisions about resources management aiming at optimizing energy efficiency, while maintaining compute environment availability and achieving desired application QoS requirements, online heuristics-based optimization algorithms are applied to solve the problem quickly in a time span up to a few seconds. Since the management of resources at each stage under investigation in this thesis comprehends two objectives, the problem can be solved following a multi-objective optimization (MOO) methodology. MOO is referred as the process of optimizing systematically and simultaneously a collection of objectives [MA04]. The aim is to search for an optimal state in the feasible decision space that maximizes the set of criteria, meaning that it is guaranteed that the solution found does not violate the given constraints. In contrast to single-objective optimization, in typical multi-objective problems there is no single global solution, and therefore it is necessary to determine a set of states that all fit a predetermined definition for an optimum. According to Abrishami et al. [ANE12], multi-objective scheduling algorithms can be addressed in three possible ways, namely: (i) finding the pareto optimal solutions, and let the user select the best solution; (ii) combination of the two functions in a single objective function; and (iii) bicriteria scheduling in which the user specifies a limitation for one criterion, and the algorithm tries to optimize the other criterion under this constraint.

The important aspects distinguishing the work presented in this thesis from the related research are the design and implementation of integrated power- and failure-aware scheduling algorithms, that implement horizontal and vertical elasticity of IaaS environments. Moreover, mechanisms to detect energy optimizing opportunities are developed, allowing to dynamically update the VMs to physical machines (PMs) mapping in order to improve energy efficiency and thus reducing operational costs. Availability is implemented through multi-objective scheduling algorithms, which autonomical and proactively tolerate faults in compute nodes. Another aspect distinguishing this work from the related research is the development of a mechanism that efficiently detects violations in diverse applications QoS requirements, based on noisy estimations. All algorithms take into account the migration overhead due to VM migrations to dynamically manage virtual environments in the cloud, in an effort to enforce applications QoS requirements. The performance analysis of the proposed solutions to tackle the issues evidenced is carried out through simulation and experiments in real cloud testbed, and using workloads, failures, and interference which characteristics follow the state-of-the-art studies in the area.

1.2 Research Problems and Objectives

The problem addressed in this thesis is twofold, namely the management of resources in cloud data centers to: (i) improve energy efficiency, where physical resources are subject to failure; and to (ii) comply with the performance required by different types of applications, with diverse QoS constraints, in energy-efficient cloud data centers. In particular, the following research problems are investigated:

- **What amount of resources to assign**. Since submitted applications to the cloud can have diverse QoS constraints (i.e., deadline or throughput), it is necessary to derive the amount

of resources necessary to comply with the application QoS constraints. By assigning the correct amount of resources to applications, wastage of resources by over-provisioning is diminished, and scarcity is avoided.

- **When to migrate VMs**. Dealing with energy inefficiencies, tolerating nodes failure, and mitigating performance deviation in applications, implies determining when active PMs are utilizing power inefficiently, or the time at which those PMs are predicted to fail, or even when the deviation in performance imposes violation of the QoS constraints. In this context, it is necessary to develop mechanisms to detect: (i) energy optimizing opportunities; (ii) nodes with insufficient reliability to satisfy applications in terms of QoS constraints; and (iii) applications executing under performance.

- **Which VMs to migrate**. Once a condition detection is made (i.e., energy inefficiency, node about to fail, or unbearable deviation in performance of VM), it is required to determine which VMs should be migrated to improve IaaS environment in terms of proposed objectives. The decision to migrate VMs must consider the overhead caused by migration in QoS requirements, the status of PMs in terms of power efficiency, reliability, and deviation in performance of hosted VMs, and the reason why migration is occurring to establish an order of migration.

- **Where to migrate the VMs selected for migration**. When a VM is marked for migrating or a new VM is submitted for scheduling, a PM must be selected to host the VM. Selecting the most appropriated PM is essential in the way it directly influences the quality of IaaS in terms of energy efficiency, dependability of virtual resources, and satisfaction of QoS constraints.

Derived by the aforementioned challenges, this thesis follows the objectives here delineated:

1. Investigate fault-tolerant mechanisms to deploy high-available virtual computing environments in the cloud, and develop and implement an energy optimizing detector that efficiently detects energy inefficiencies in IaaS environments.

2. Design an online scheduling algorithm to create power- and failure-aware IaaS environments.

3. Investigate performance interference causes and detectors in co-hosted VMs, and design and implement a performance deviation estimator that determines, for different QoS constraints, the differential in resources needed to satisfy the QoS requirements.

4. Design an online scheduling algorithm to create interference- and power-aware IaaS environments.

1.3 Evaluation Methodology

The mechanisms proposed in this thesis are evaluated through discrete-event simulation and experiments in real cloud testbed created for this purpose. In a first attempt, simulation approach is essential to deal with the extremely complex process of evaluating the efficacy of different mechanisms on large-scale virtualized data center production environments, where it is hard to guarantee repeatable conditions for such a set of experiments. Simulation provides the ability to ensure the reproducibility and repeatability of experiments, which can be easily conducted with different parameters to examine the behaviour of the proposed mechanisms in diverse circumstances. Therefore, discrete-event simulation has been chosen as a first step, towards creating cloud computing environments and deploying views of infinite computing resources, so as to evaluate the performance of the proposed mechanisms. In a second moment, tests are done in a real platform in order to provide preliminary results of the adoption of the proposed algorithms.

The application workloads to use in simulations and experiments in real cloud testbed, follow the last version of the Google cloud tracelogs [MGTX13]. The data collected from Google data centers spans for over 29 days, yielding significant data on the characteristics of submitted workloads and the management of cluster machines. These studies enable further work on important issues, such as resource optimization, energy efficiency improvements, and failure correlation. In turn, the failure characteristics describing nodes reliability are modelled based on failure data collected at two large high-performance computing sites [SG10]. The first data set has been collected at LANL and spans over nine years. The second data set has been collected on one large supercomputing system comprising 20 nodes and more than 10,000 processors, over the period of one year. Regarding performance deviation, the attributes of interference occurring among applications during runtime follow the characteristics obtained from the in-depth analysis of the state-of-the-art in this topic.

In order to consistently evaluate the performance of the mechanisms proposed in this thesis, state-of-the-art scheduling algorithms are implemented for comparison purposes.

1.4 Summary of Contributions

Considering the aforementioned objectives, the major contributions of this thesis are listed as follows:

1. **The development of power- and failure-aware scheduling algorithms that implement vertical and horizontal platform elasticity**. This thesis proposes two improved scheduling algorithms to construct power- and failure-aware IaaS cloud environments, aiming at providing to end-users a high-available and reliable computing environment, while the energy consumed is reduced, with a limited application performance impact.

2. **The development of a mechanism to reactively detect energy optimizing opportunities to assist in creating and managing power- and failure-aware virtual clusters**. This

thesis proposes a threshold-based power efficiency optimizing mechanism that, based on PMs statuses, detects under-loaded PMs that can be switched to low power modes in order to optimize the energy efficiency of the whole cloud system.

3. **An extensive evaluation of the energy optimizing mechanism and scheduling algorithms with randomly generated workloads and with workloads that follow the latest version of the Google cloud tracelogs**. The proposed mechanisms in this thesis are tested with random- and Google cloud tracelogs-based workloads, through simulation and experiments in real cloud testbed.

4. **A dynamic configuration of the CPU portion assigned to each VM that reduces the consumed energy and maintains the service-level performance**. Unlike state-of-the-art scheduling algorithms implemented in cloud middle-ware stacks such as OpenStack [Jac12], the proposed power- and failure-aware scheduling algorithm implements horizontal and vertical elasticity of resources. For that, the algorithm leverages tools presented in state-of-the-art virtualization technology, such as cap from the Xen credit scheduler.

5. **A mechanism to detect and estimate the performance deviation in CPU-bound workloads from the QoS requirements, even when performance data samples are noisy**. This thesis proposes a mechanism to deal with performance deviation in CPU-bound tasks due to contention in on-chip resources, by estimating the slowdown occurring in tasks runtime, and assigning resources accordingly to assure that the tasks are executed by their deadlines.

6. **A strategy for the enforcement of QoS requirements of network-bound applications which workloads are bursty in time**. In order to deal with oscillation in demand of resources by network-bound applications, this thesis proposes a strategy that detects performance deviations in the number of requests served per time unit, and dynamically defines soft limits accordingly during application runtime. In this regarding, the strategy assures application QoS requirements by avoiding scarcity of resources, and it diminishes the waste of resources at the same time.

7. **An algorithm for scheduling of heterogeneous workloads in virtualized data centers, with different characteristics and specific SLA requirements**. This thesis proposes an improved power- and interference-aware scheduling algorithm, that performs vertical and horizontal elasticity, to construct interference- and power-aware virtual clusters (i.e., IaaS cloud environments) to execute the tasks under the QoS requirements.

8. **A study about the trade-off between the two opposite objectives: energy efficiency versus performance improvement**. An extensive set of simulations is carried out in order to assess the performance of improved power- and interference-aware algorithm in terms of construction of power-efficient vs high-performance clusters. To this end, a control parameter in the algorithm was tuned, and the results properly discussed.

1.5 Thesis Organization

The core chapters of this thesis are structured as listed below, and are derived from the set of research papers published during the course of the PhD candidature. The remainder of this thesis is organized as follows:

- Chapter 2 presents a literature review on the current related work in IaaS management, introduces a comprehensive analyses of virtualization capabilities and limitations in enhancing IaaS management, and positions this thesis in regards to those works.
- Chapter 3 introduces an overview of power and energy concepts, analyses and identifies the sources of power consumption in a data center, and follows with a taxonomy and survey of fault tolerance schemes in computing systems. The chapter introduces the current related work as well, and positions this thesis in regards to those works.
- Chapter 4 presents a energy optimization mechanism, which detects energy optimizing opportunities, and two scheduling algorithms that create energy-efficient and high-available IaaS cloud environments. This chapter is derived from [SB13b, SB13c, SB14c]:
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "Dynamic power- and failure-aware cloud resources allocation for sets of independent tasks.", in IEEE International Conference on Cloud Engineering (IC2E), San Francisco, 2013.
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "Optimizing energy-efficiency in high-available scientific cloud environments.", in 3rd IEEE International Conference on Cloud and Green Computing (CGC), Karlsruhe, Germany, 2013.
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "Towards high-available and energy-efficient virtual computing environments in the cloud.", Future Generation Computer Systems, Elsevier, vol. 40, pp. 30–43, 2014.
- Chapter 5 presents an approach to detect and mitigate the performance deviations in two different types of applications, with specific QoS constraints, and it proposes a scheduling algorithm to create interference- and power-aware IaaS cloud environments. This chapter is derived from [SB13a, SB14a, SB14b]:
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "Last-Level Cache Interference-Aware Scheduling in Scientific Clouds", Parallel & Cloud Computing, vol. 2, n. 4, pp. 116–125, Oct. 2013.
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "Estimating Effective Slowdown of Tasks in Energy-Aware Clouds.", in 12th IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA), Milan, 2014.
  - **Sampaio, Altino M.**, and Jorge G. Barbosa, "A Performance Enforcement Mechanism for Energy- and Failure-Aware Cloud Systems.", in 5th IEEE International Green Computing Conference (IGCC), Dallas, 2014.

  - **Sampaio, Altino M.**, Jorge G. Barbosa, and Radu Prodan, "PIASA: a Power and Interference Aware Resource Management Strategy for Heterogeneous Workloads in Cloud Data Centers.", Simulation Modelling Practice and Theory, submitted in February, 2015.

- Chapter 6 concludes the thesis by summarising the main findings and contributions, and by discussing the future research directions.

Chapter 2

Virtualization-based Resource Provisioning in Modern Data Centers

Virtualization is extensively applied in cloud computing data centers to implement the vision of IaaS, in which IT infrastructure is deployed in a provider's data center as VMs. Users' applications are packaged within VMs, which addresses significant benefits for cloud providers in reconfiguring the cloud infrastructure to achieve predefined goals. Today's virtualization technologies enable virtualization of many factors, such as IT resources, hardware, software, operating systems and storage, whereas the execution environment is decoupled from the physical platform. The aim of this chapter is to provide a view of IaaS management, and a comprehensive analyses of virtualization software capabilities and limitations in enhancing IaaS management, and positioning this thesis in regards to those works.

2.1 Introduction

MODERN society increasingly depends on large-scale distributed systems to conduct business, government, and defence. A distributed system is essentially a collection of independent computers, sharing resources and interacting with each other towards achieving a common goal, via a computer network [TVS02]. From the users' point of view, a distributed system appears as a single coherent system. The shared resources in a distributed system include data, storage capacity, and computational power, and the distributed computing infrastructure can be used to serve diverse objectives, ranging from executing resource-intensive applications (e.g., CPU-bound applications) to serving scalable Internet applications (e.g., CPU- and network-bound applications). In particular, in the last recent years, large-scale distributed systems have been consolidated itself as a very powerful platform to develop and operate a new generation of business, delivered through the Internet. No matter the size or location of the business, the Internet allows almost any business to reach a very large market. And so, distributed systems have become a powerful mean to support the evolution of civilizations, providing services for business, development of research

and science, and welfare of populations. However, articulation of provided services and deployed resources is challenging, making the integration of decentralized services and resources difficult.

Recently, an increasingly interest in Service Oriented Architectures (SOA) [Erl05] and constantly growing solutions for virtualization have leading to the concept of cloud. Cloud computing [AFG+10] can be defined as a specialized distributed computing paradigm that utilizes networked computing systems to handle data and computation seamlessly. It represents a new type of computational model, providing better use of distributed resources, while offering dynamic, flexible infrastructures and QoS guarantees. Key concepts distinguishing cloud computing from other paradigms includes on-demand, high-availability, and high-scalability access to large-scale distributed resources, with Service Level Agreement (SLA)-driven management [BYV+09]. From a hardware point of view, users have the illusion of infinite computing resources that are available on demand. By using cloud based systems, costumers can have easy access to large distributed infrastructures, with the ability to scale up and down the assigned computing resources according to their applications needs. Such ability represents an enormous advantage compared to other computing paradigms since cloud users can concentrate specifically in developing their work, without concerning about over- or under-provisioning of resources. What is more, cloud users are free from the burden of requiring the large capital outlays in hardware to deploy their applications or the human expense to operate it, because cloud manages it transparently to costumers. Another important feature of cloud is that organizations pay for computing resources on a consumption basis, very much like the utility companies charge for basic utilities such as electricity, potentially leading to cost savings.

According to Schubert et al. [SJNL10], clouds can be classified according to their service types and deployment modes.

- **Infrastructure as a Service (IaaS)**: It provides resources, that have been virtualized, as services to the user, such as virtual computers, meeting various computing needs, and even virtual clusters. Cloud infrastructures apply resource elasticity and server consolidation techniques to rapidly adapt to environment changes (i.e., variation in computational capacity needs, physical nodes availability, etc.), and provide on-demand access to computational resources in a pay-as-you-go basis. Common operation in clouds implies that provisioning of services are specified and regulated using SLAs, and requests for instantiation of services may be accepted or rejected based on current and predicted infrastructure load and capacity. Among public IaaS providers, Amazon EC2[1] offers access to EC2 instances which looks much like physical hardware, letting users control the software from the kernel upwards. However, this approach has inherent difficulty supporting automatic scalability and failover, since traditional methods, such as replication, are highly application-dependent.

- **Platform as a Service (PaaS)**: It provides developers with an environment upon which applications and services can be developed and hosted. Typically, users control the behaviour

[1] Amazon EC2. http://aws.amazon.com/ec2/

and execution of a server hosting engine through dedicated Application Programming Interface (API) deployed by cloud providers. In this way, developers can concentrate in creating services instead of worrying about building an environment for running them.

- **Software as a Service (SaaS)**: It provides access to software packages / services using the cloud infrastructure or platform, but keeping users unaware of underlying infrastructure. SaaS allows end-users to avoid rigid license agreements or infrastructure costs.

Furthermore, clouds can be classified based on the business model of the provider as:

- **Private Clouds**: Typically owned by the respective organization, services deployed can be accessed and managed exclusively by people of the organization, thus not being directly exposed to the customer.
- **Public Clouds**: Organizations may offer their own cloud services to clients or other service providers, thus being available to general public. In this context, enterprises can utilize such cloud providers services to avoid build up their own infrastructure with consequent cost reduction.
- **Hybrid Clouds**: It consists of mixing private and public cloud infrastructures and services. This model allows enterprises to keep control over sensitive data by employing local private clouds, and to outsource parts of their infrastructure to public clouds so as to achieve a maximum of cost reduction.

Additional classifications can eventually be found in literature, or at different granularity. This thesis addresses the problem of IaaS resources management in a private cloud, where users submit applications, which may have different types of workloads and diverse QoS requirements.

2.2 Virtualization Technology

Virtualization [VNE+08] is the main technology in cloud computing to deploy IaaS. It promises a reduction in cost and complexity through the abstraction of computing resources such that a single PM is able to support a large number of disparate applications running simultaneously. In turn, each application runs in a logical container – the VM. In this sense, virtualization enables a reduction in the number of physical components, meaning fewer assets to configure and monitor, and consequent reduction in complexity and cost of managing cloud infrastructures.

2.2.1 Concept

The concept of virtualization was first introduced with the IBM mainframe systems in the 1960s, to refer to a virtual machine [Ada66]. A virtualization system is an architecture able to separate and isolate an operating system from the underlying hardware resources and lower level functionalities (e.g., servers, network links, and host bus adapters). Server virtualization [Gol74] is performed

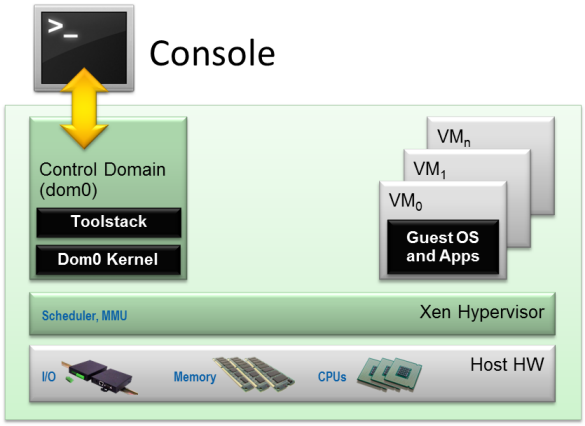


Figure 2.1: Virtualization concept[2].

on a given hardware platform by introducing a thin hypervisor layer, sometimes called virtual machine monitor (VMM), which sits on top of the physical hardware resources and creates a simulated computer environment (i.e., a VM).

Hypervisors can be implemented in server firmware or in software [LDL+08]. Virtualization splits a single physical computer into multiple logical computers, or VMs, each accommodating its own guest software, as described in Figure 2.1. Then, the hypervisor multiplexes and arbitrates access to resources of the host platform so that they can be shared efficiently among multiple VMs. The guests run just as if they were installed on a stand-alone hardware platform. The number of VMs per PM is limited by the host hardware capability, such as core number, CPU power, RAM resources, and can be defined as well by the software provider.

According to the type of virtualization, there is no requirement for a guest operating system (GOS) to be the same as the physical host one. Guests are able to access specific hardware components, such as hard disk drives, network interfaces, etc.

2.2.2 Classification of Virtualization

In the x86 CPU architecture, hierarchical protection domains, also called protection rings, are a mechanism to manage access to the computer hardware with different privileges. Rings are arranged in a hierarchy from most privileged (i.e., the ring 0) to least privileged (i.e., the ring 3). The kernel of an OS executes its instructions at the most privileged level, the ring 0, to have direct access to the memory and hardware. The virtualization software is placed under the OS to be able to run in ring 0 and to directly access the hardware, forcing the OSs to move to a level ring with less privilege than the VMM executing in ring 0. In this regarding, VMMs have to implement a mechanism to support privileged operations in OSs hosted in VMs and no more executing in ring 0. For that, three major alternative architectures exist for handling sensitive and privileged instructions [VMw07], namely:

[1] http://wiki.xen.org/wiki/Xen_Overview

- **Full Virtualization**: The guest OS needs no modification, because it is completely decoupled from the hardware and is unaware that it is being virtualized. The hypervisor translates OS instructions on the fly into instructions that have the intended effect on the virtual hardware. Instead, user level instructions are directly executed on the processor to improve the performance. All requests to access the hardware are handled on demand by the hypervisor, which caches the results for future use. This alternative offers the best security and flexibility, allowing a higher number of operating systems to be virtualized.

- **Para-virtualization**: In this mode, the hypervisor layer runs directly over the hardware, which provides the resources needed by the virtual machines. Then, a privileged operating system instance runs over the hypervisor, and is responsible for controlling it, and for managing virtual machines. The hypervisor provides hypercall interfaces to be used by guest operating systems. This means that guest operating systems are modified, in order to replace the privileged operations with hypervisor calls, in such a way that they know that are being virtualized. Para-virtualization environments impose a low virtualization overhead and provides optimized operations, but their performance can vary greatly depending on the workload. Examples of para-virtualization systems are Xen[3] and VMware Infrastructure[4].

- **Hardware Assisted**: This alternative implies hardware support such as Intel VT-x[5] and AMD AMD-V[6] based machines, present in systems since 2006. In this technology, the VMM runs in a new root mode below ring 0, meaning that specific CPU calls are not translated by the hypervisor, and are sent directly to the CPU. However, this approach introduces a high hypervisor to guest transition overhead.

2.2.3 Virtualization Features

The recently resurgence of virtualization relates to the possibility to deal with issues in the data center, such as variance in demanding requirements of computing resources, power inefficiency due to hardware underutilization, high system administration costs, and reliability and high-availability. Current virtualization technologies provide powerful mechanisms to tackle these issues, namely VMs resizing, migration and checkpointing. VM resizing mechanism permits the adjustment of resources allocation, by adding additional CPUs, network interfaces, or memory. In turn, VM migration [CFH+05] mechanism gives the ability to easily move VMs from one physical node to another one using live or stop and copy migration alternatives. Both VM resizing and migration techniques are the necessary conditions to achieve elasticity and the appearance of infinite capacity available on demand. At the same time, virtualization benefits cost reduction by server consolidation (i.e., aggregation of existing VMs in less quantity of PM, hence decreasing the number of active PMs), since the number of hosting servers can be minimized based on better

[3] http://www.xen.org
[4] http://www.vmware.com
[5] htto://www.intel.com/go/virtualization
[6] http://www.amd.com/us/solutions/servers/virtualization/Pages/virtualization.aspx

utilization of the expensive hardware resources. Therefore, through resource elasticity and server consolidation techniques, cloud infrastructures are able to rapidly adapt to energy inefficiencies and failures in physical nodes, providing high-availability access to computational resources on demand. Thus, VMs resizing and migration have emerged nowadays as promising techniques to be utilized by resource management algorithms since they can cope with resources allocation problems in modern data centers. For example, based on these techniques, Lagar-Cavilla et al. [LCWS+09] developed the SnowFlock system, a VM fork approach that rapidly clones a VM into several replicas running on different PMs. Similiar to [HUM+11], it uses fast VM instantiation by initially copying and transmitting only the critical metadata (authors called it VM descriptor) necessary to start execution on remote host. Memory-on-demand mechanism fetches portions of VM state, from an immutable copy of the VM's memory from the time of cloning, over the network as it is accessed. Cooperatively, avoidance heuristics algorithms try to reduce superfluous memory transfers that will be immediately overwritten, and multicast replies to memory page requests. These key techniques enable SnowFlock to introduce little runtime overhead and small consumption of I/O resources, leading to good scalability. VMs disks are implemented with blocktap driver. SnowFlock represents an example of how modern data centers can leverage virtualization to provide excess load handling, opportunistic job placement or even parallel computing, as well as to provide fault tolerance due to replication of VM instances.

Nowadays, there are several system-level virtualization solutions offered by some commercial companies and open-source projects. Some of the major virtualization software include Xen[7], KVM[8], and VMware[9]. The next subsection discusses one of the most popular virtualization technology solution – Xen hypervisor, which supports the management mechanisms here described.

2.2.4 The Case of Xen Hypervisor

The Xen hypervisor is an open-source high performance resource-managed hypervisor [BDF+03]. Xen was initially created by the University of Cambridge Computer Laboratory and is now developed and maintained collaboratively by the Xen community and engineers from several innovative data center solution vendors. The software is available at no charge in both source and object formats, and it is released under the GNU General Public License (GPL2).

Xen is classified as a "bare-metal" hypervisor because it runs directly on top of the physical machine in the most privileged processor-level. The hypervisor is the first program running after exiting the bootloader, and it is responsible for handling CPU, Memory, and interrupts. On top of that, Xen can execute a number of virtual machines, also known as domains or guests. There can be two types of domains, namely: (i) the privileged domain known as Dom0 or driver domain; (ii) and the unprivileged domains called DomUs. The Xen Dom0 is a special domain that is responsible for controlling the hypervisor and managing virtual machine (i.e., DomUs) creation, destruction, and configuration. For that, Dom0 runs a modified version of operating system that uses native

[7] http://www.xensource.com
[8] http://www.linux-kvm.org
[9] http://www.vmware.com

drivers to manage all the physical devices in the system. In turn, domains DomUs communicate with Dom0 by generating traps into it, and can also use hypercalls to invoke functions in the hypervisor. Xen hypervisor virtualizes the physical CPUs, providing virtual CPUs (VCPUs), on which the VMs run. This software supports para-virtualization and hardware assisted modes of virtualization.

Xen currently supports various scheduling algorithms to schedule domains, some of them were extensively analysed to determine their performance considering different scheduling parameters [CGV07, XSWJ08, SC08]. Xen CPU credit scheduler[10] is the default scheduler in Xen since version 3.0, and is designed to ensure that each VM and/or a VCPU gets a fair share of the physical CPU resource. In the credit scheduler, to each domain is given the opportunity to configure CPU affinity and priority, by means of pinning VCPUs to specific cores and defining cap and weight scheduler parameters, respectively. The cap scheduler parameter specifies the maximum percentage of CPU resources that a VM can get, while weight parameter determines the number of credits associated with the VM, which controls the priority in the execution of VMs. According to those weights (e.g., number of credits), the scheduler assigns physical CPU time allocation among all the VMs. For example, Xen credit scheduler can allow execution of two tasks, one VM receiving 30% of total CPU capacity, and the other the 70% remaining. In this regarding, there are examples of capping use to adjust dynamically resource allocation based on task usage, and to make sure co-located tasks cannot consume more resources than those allocated to them [SSGW11]. This thesis leverages the cap parameter feature in scheduling applications.

Xen supports migration of VM instances from one PM to another. It works by transferring the entire memory state of the kernel, all processes, and all application states, of the VM to be moved to another PM. Live migration and stop-and-copy migration are the two forms of migration supported by Xen hypervisor [CFH+05]. Stop-and-copy migration methodology involves suspending the original VM, copying all pages across to the destination VM, and then starting the new VM in the final PM. The downtime and total migration time are proportional to the amount of physical memory allocated to the VM, and depends on the free bandwidth at the moment of migration [Fu10, MCTL+14]. In turn, live migrating VMs between physical nodes implies to iteratively copy the memory of the VM to the destination without stopping its execution [CFH+05].

Fu et al. [Fu10] investigated the performance of both migration methods in fault tolerance systems and concluded that stop-and-copy migration is simpler and faster to accomplish, despite introducing superior service downtime. In turn, live migration imposes shorter service downtime, but the overall migration time is longer. In this regard, this thesis leverages stop-and-copy migrations method because they are more preferable in failure-aware resource management schemes due to small overhead and the imperfection of state-of-the-art failure prediction techniques [FX07a]. Yet in the context of fault tolerance, Xen implements a checkpoint mechanism called Remus [CLM+08], which uses copy-on-write (CoW) techniques [YAB+86] to asynchronously replicate/checkpoint the primary VM memory and disk to the backup VM in a very high frequency. The backup VM maintains a pre-copied disk image file to receive the

[10] http://wiki.xen.org/wiki/Credit_Scheduler

updates. Unfortunately, these pre-copies operations are very time consuming and often prove counter-effective [Phi05]. Fault tolerance approach based on checkpointing/restart mechanisms is out of the scope of this thesis, in favour of lighter alternatives in terms of performance overhead. In this investigation work, Xen is the selected hypervisor followed throughout the research to implement proposed mechanisms.

2.2.5 Virtualization-based QoS in Modern Data Centers

The current service oriented economy trend [AD09], supported by distributed computing paradigms such as cloud computing, has led to an increased concern about aspects regarding the quality and reliability of the services offered. As consumers move towards adopting such SOAs, the exploitation of service oriented technologies is followed by a growing interest of QoS mechanisms [SS09]. However, providing QoS guarantees to users is a very difficult and complex task, because the demands of the consumers' service vary significantly, and initially assigned computing resources to applications tend to not perform as expected as they are being shared among different types of workloads. In this regarding, services to costumers are often provided within the context of a SLA, which ensures the desired QoS. SLAs can be defined as a static mechanism of agreements between users and service providers, in which all expectations and obligations of a service are explicitly defined [PRS09]. The SLA's main objective is to serve as the foundation for the expected level of service between the consumer and the provider, translating users' expectations in terms of QoS into infrastructure level requirements. This way, users define their QoS requirements in high-level SLA metrics, which in turn correspond to low-level resource metrics used by cloud management system to enforce service utilization policies and usage commitments. The QoS attributes are generally part of an SLA, such as response time, throughput, and maximum execution time to accomplish a task, and need to be closely monitored so the management system can guarantee the QoS attributes on each application execution [KL03].

Virtualization technology is useful for cloud providers, in assisting the management of IaaS to fulfil users' applications QoS requirements. The correct and effective use of the features provided by virtualization, such as migration and Xen credit scheduler parameters, allows cloud providers to cope with important current data centers issues, such as fault tolerance and energy consumption (see Chapter 3). Despite the advantages of using virtualization, there are unfortunately some limitations related with the use of the technology as well. Next, technological limitations are discussed.

2.3 Technological Limitations of Virtualization

Nowadays, cloud providers are faced with new challenges such as reduction of power consumption and guaranteeing SLAs, the key element to support and empower QoS in these environments. However, guaranteeing the expected QoS for applications, and reducing the energy consumption at the same time, is not trivial due to QoS deviations. In the base of such deviations from expected

performance and promised resources availability and capacity lies technological limitations and diverse scheduling policies. Regarding the former reason, modern virtualization technologies do not guarantee effective performance isolation between VMs, meaning that the performance of applications can change due to the existence of other co-resident VMs sharing the underlying hardware resources (e.g., [CKK11, MLPS10, SC09, ZT12]). In fact, a task running in the same VM on the same hardware but at different times will perform disparately based on the work performed by other VMs on that physical host. This phenomenon has been studied, and is known as performance interference. The later reason involves cloud providers policies, regarding achieving specific objectives such as reduction of operational costs. To this end, cloud providers try to schedule multiple VMs onto fewer servers (i.e., consolidation of VMs) in order to improve resource utilization, and to reduce power consumption. However, optimizing energy efficiency through consolidation of VMs is challenging since it can lead to additional degradation in applications' performance. The counter effect of consolidation can be even more severe depending on the type of workload a VM is running. For example, in the case of web applications, with bursty workloads, the demand of resources can lead to scarcity of resources, and consequent SLA violations. In the case of batch jobs, consolidation can produce severe slowdown due to contention in on-chip resources, meaning that workloads take much longer to complete under consolidation than without it, despite the advantages in energy savings [BZF10].

Dealing with performance deviations is a complex task, because some resources are very hard to isolate, such as on-chip shared resources (e.g., cache memory, memory buses), and because of bursty oscillation in demand of resources (e.g., I/O network). Research literature on this subject analyses diverse causes of performance degradation due to contention in shared resources [HL13, ZT12, PLM+13]. Next, the sources of performance interference, considered in this thesis, are presented.

2.3.1 Last-Level Cache

Current multi-core processors are equipped with a hierarchy of caches. Commonly, a CPU chip has one or more cache levels L1 / L2 dedicated to each of its cores, and a single shared last-level cache (LLC) L3, such as the Intel Nehalen Core i7. For most processors, data is inserted and evicted from the cache hierarchy at the granularity of a cache line of 64 Bytes of size [GLKS11]. If the co-hosted applications total working set size exceeds the PMs CPU LLC size, then applications will definitely degrade in performance [MTS+12]. Because presently it is not possible to measure the cache usage directly, the hypervisor can measure the impact of co-hosted VMs in one another performances by tracking the misses per instruction, collected through performance counters [DFB+12].

2.3.2 I/O Network

Network interface cards (NICs) can also affect the performance of co-hosted VMs. For security reasons, Xen hypervisor imposes that VMs perform I/O operations through special device located

at privileged domain Dom0. Each VM uses a virtual NIC (VNIC) that allows exchanging data with Dom0, by sharing memory pages. The notification of VMs is done based on virtual interrupts. High number of received/transmitted packets to/from VMs, as well as small fragmented packets, generate high rate of hypervisor interrupts and data multiplexing to the corresponding VMs. High rate of multiplexing/demultiplexing of packets among Dom0 and VMs creates a communication bottleneck and causes interferences among VMs, such as packet loss for high latency, fragmentations and increased data copying overhead [PLM+10].

2.3.3 CPU Scheduling Policies

While cloud providers try to reduce operational costs by improving utilization of resources by means of consolidation, from the consumer point-of-view it is essential that the cloud provider offers guarantees about service delivery. However, to deal with these two opposite objectives is challenging, because while the former's objective is essential for profit, the second is crucial for the trust of the service perceived by users. Some techniques, such as cap parameter in Xen credit scheduler (see Section 2.2.4) and Linux traffic controller (tc) [HGM+02], enforce the isolation of execution environments by imposing limits in the usage of resources. However, because some application workloads, such as web workloads, are bursty and vary according to the number of users accessing the applications, such limits must be updated dynamically and effectively to avoid scarcity and waste of resources. While scarcity of resources lead to SLA violations, waste increases unnecessary energy consumption.

2.4 Performance-aware Scheduling

A large volume of research has been done in the area of performance interference modelling and scheduling, due to contention for multiple resources in a system. The efforts represent a great contribute because they offer deeper understanding of the key factors for effective resource sharing among applications running in virtualized cloud environments. Next, the research efforts in interference analysis and QoS-aware scheduling are presented.

2.4.1 Interference Analysis and Modelling

Notable research work has contributed with models to capture performance interference effects among consolidated workloads. The extent of degradation clearly depends on the combination of applications that are co-hosted, shared resources, and even the kind of virtualization mode. Therefore, determining the contribution of all these parameters to the interference level on application QoS is critical.

Somani et al. [SC09] conducted research, by running combined benchmarks to give an insight about the isolation properties of Xen hypervisor relating to CPU, network bandwidth, and disk I/O speed resources. The results showed that Xen credit scheduler behaves well when running two non-similar resource intensive applications. Specifically, it provides good isolation when running

high throughput and non-real time applications, but it becomes difficult to predict the performance and the time guarantees when running soft real time applications. These insights contribute to place applications in the data center in order to get the maximum isolation and fairness. In a later work [SC10], the authors studied the effect on performance interference from different CPU scheduling configurations in Xen hypervisor, for combined I/O and CPU intensive applications. Findings reveal that Xen credit scheduler is relatively good in the case of dedicated CPUs, which can be achieved by means of assigning the virtual CPUs (VCPUs) to physical cores (called core pinning or core affinity). For non-pinning configurations, credit scheduler shows much higher time to complete the tests, as compared to the pinned case. This is due to the opposite performance impact of unwanted load balancing which results into more VCPU switches and cache warming as co-hosted domains are doing heavy disk I/O. In running CPU intensive applications, credit scheduler proved itself good, and even showed better performance than Simple Earliest Deadline First (SEDF) scheduler when running heavy I/O (net and disk). For interactive applications, such as games and office applications, SEDF scheduler showed good average latency, with minimum latency similar to credit scheduler. The credit scheduler can, in some cases, result into higher peaks of latency for CPU intensive applications, because of its less prompt service behaviour.

Koh et al. [KKB+07] studied the effects of performance interference by co-locating combined sample applications, chosen from a set of benchmark and real-world workloads. Those workloads comprehended different characteristics in terms of CPU, cache, and I/O utilization. Then, authors collected the performance metrics and runtime characteristics for subsequent analysis and identified applications combinations that generate certain types of performance interference. For example, some findings indicate that there is a significant degree of performance interference between I/O-intensive applications, and memory-intensive applications can suffer a performance degradation from 20-30%. On the other hand, CPU-intensive programs are able to achieve better performance with I/O-intensive applications. Xen was the virtualization technology used. Authors also proposed a weighted mean method to predict the expected performance of applications based on their workload characteristics, with a 5% average error. A later work [PLM+10] extended the study to performance interference among VMs, but in the context of network I/O workloads that are either CPU-bound or network-bound. Authors pointed out interesting findings about combination of types of workloads, namely: (i) network-intensive workloads can lead to high overheads due to extensive context switches and events in Dom0; (ii) CPU-intensive workloads can incur high CPU contention due to the demand for fast memory pages exchanges in I/O channel; (iii) CPU-intensive workloads combined with network-intensive workloads incurs the least resource contention, delivering higher aggregate performance. Considering that experiments were taken using Xen hypervisor, authors concluded additionally that the efficiency of Dom0 highly affects the performance of multiple VMs.

Govindan et al. [GLKS11] proposed a technique for predicting performance degradation among consolidated workloads (not I/O intensive) due to shared last-level processor cache space and memory bandwidth. Performance degradation is measured in terms of increase in task finish time compared to its finish time when running alone with the same resource allocation. In

this regarding, authors developed Cuanta, which implements a synthetic cache loader benchmark to create a cache clone for each application (i.e., VM), which is later used for predicting the performance degradation for different application co-location scenarios. The cache loader benchmark can be tuned in order to generate fine-grained cache access patterns at the granularity of the available sets and ways of modern set-associative caches. Cuanta implies a profiling phase, by conducting a set of experiments to generate a full interference vector for an application, when executed with any set of $N-1$ applications co-located on the other cores. Authors underlined the portability of the proposal, since it does not rely on cache-related hardware counters, and the needless of changing the software stack of the hosting platforms. Results demonstrated that Cuanta is able of predicting application performance for a variety of co-location scenarios within 7% of the measured performance. Authors have also illustrated the usefulness of Cuanta for guiding VM consolidation decisions to yield better workload placement decisions for given performance and resource cost objectives, finding the appropriate energy efficiency and performance trade-off.

Dwyer et al. [DFB+12] contributed with a tool that takes as inputs performance counter values obtained on a consolidated workload, and estimates the performance degradation an application is suffering relative to running on the CPU alone. The output of the tool is a measure of the slowdown in the execution of the CPU-bound application. The slowdown is caused by performance interference among co-hosted workloads that share LLC resource. The tool applies machine learning to determine the degradation online, on previously unseen workloads, though it is trained offline using a set of widely available benchmark programs. Authors reported that the tool is able to determine the degradation within 16% of the true value on average, and without perturbing the execution of applications.

Lim et al. [LHK+12] proposed D-factor model, which models the problem of performance interference within a physical server based on a collection of multiple resource-queues, which contention creates non-linear dilation factors of jobs. Authors tested CPU- and disk I/O-bound applications. Key findings revealed (i) multiple-resource contention creates non-linear dilation factors of jobs; (ii) linear relationship between total completion time and individual completion times for the same type of jobs; (iii) co-location of workloads utilizing different system resources leads to best efficiency. Authors claim that proposed D-factor model is able to predict the completion time of coexisting workloads within a 16% error, for realistic workloads.

Verma et al. [VAN08a] performed an experimental study about application performance isolation, virtualization overhead with multiple VMs, and scenarios where applications were isolated from each other. From the insights obtained, authors proposed a framework and methodology for power-aware application placement for high-performance computing (HPC) applications, which considered both CPU and cache size constraints. Some of those important insights include: (i) isolation on virtualized platforms works for homogeneous workloads (in terms of size) as long as they are very small, and the sum of them is inferior than cache size, or very large (e.g., 60 Mbytes); (ii) performance tends to be better when the number of VMs is a power of 2, because memory banks as well as cache segments are typically power of 2; (iii) the performance of an application degrades as soon it gets closer to the size of the cache resource, and stabilizes in some

point after; (iv) the power drawn increases at point of sum of applications working set equalling the cache size, and then starts to decrease, since the power increase in memory access cannot compensate the decrease in power drawn by the CPU caused by the decreased throughput (instructions dispatched per second). This last finding was confirmed later by [MYXW13]. However, authors work is based only on contention over CPU and LLC resources, and the approach does not consider auto-scaling requirements of applications, nor readjusts the provisioning of resources in the case of non observation of applications' QoS requirements.

2.4.2 QoS-aware Resource Management

The extensive research in understanding and modelling performance interference effects among consolidated workloads has shed significant light on the kind of settings and combinations that most degrade applications performance. Based on the outcomes from these studies, several authors have proposed alternative QoS- or performance-aware scheduling algorithms in order to achieve distinct objectives, and guaranteeing applications required QoS at the same time.

In this regarding, Kim et al. [KEY12] proposed a Performance-Maximizing VM (PMV) scheduler algorithm that deals with performance impact due to contention in a shared LLC. Authors showed that the performance degradation can reach almost 43% due to interference in shared last level cache and memory bus. The aim was to identify the VM arrangement that minimizes the performance interferences and the number of active physical servers, thus being possible to reduce energy consumption by turning off redundant servers. The proposal implies to co-locate a VM that has large cache demand with a VM that has small cache demand, in order to improve overall performance. Cache demand for a VM is inferred from LLC reference ratio, and can be determined when VM is running, without the need of prior profiling of workloads. The scheduling algorithm follows this strategy, allocating VMs with the largest LLC reference ratio in PMs in which the sum of LLC reference ratios is the smallest. Results have demonstrated a decrease in average performance degradation ratio in about 16%. Even though, average performance degradation ratio remains above 14%.

Zhu et al. [ZZA10] developed pSciMapper to consolidate scientific workloads with dissimilar requirements, focused on the interference between the resource requirements, and reduction of total power consumption with lower execution slowdown. Authors started by investigating how the resource usage impacted the total power consumption, and then they analysed the correlation between performance and power consumption with consolidation of different types of workloads. Important insights from results were observed, namely: i) CPU utilization represents the largest impact on total power consumption (though other resource activities, such as memory footprint and cache activity, also impact power consumption); ii) Disk and network I/O add a constant power consumption; iii) Virtualization (Xen) induces very small additional overhead, and dynamic cap set in Xen credit scheduler saves power comparing to using work-conserving mode; and iv) consolidation of workloads with dissimilar requirements can reduce the total power consumption and resource requirements significantly, without substantial degradation in performance. Based on these observations gained from experiments, the consolidation algorithm estimates the consumed

power and execution time using a Kernel Canonical Correlation Analysis (KCCA) [BJ03] model trained offline, and apply online consolidation algorithm using hierarchical clustering [Joh67] and the Nelder-Mead optimization algorithm [NM65], to search for the optimal VMs to PMs consolidation mapping (i.e., power requirements is minimized, with small impact on performance). Results demonstrated that PsciMapper is able to reduce power consumption by up to 56%, with less than 15% slowdown, with low scheduling overhead.

Chiang et al. [CH11] proposed Tracon framework so as to mitigate the interference effects from co-located data-intensive applications, and thus improving the overall system performance. Authors focus on a specific I/O type of resources. Tracon is composed of three major components, namely: i) interference prediction component to infer application's performance from resources consumption; ii) an interference-aware scheduler to utilize the first component and assign tasks to physical resources in an optimized way; iii) and task and resource monitors to collect application characteristics to feed both previous components. The framework works based on modelling and control techniques from statistical machine learning, for reasoning about the application's performance under interference to manage the virtual environment. In this way, the application performance is inferred from an interference prediction model which is based on the resource usage. Authors test three modelling approaches, and three scheduling algorithms. Results showed that Tracon can improve application runtime and I/O throughput up to 50% and 80%, respectively.

Mey et al. [MLPS10] conducted in-depth measurement-based studies about performance impact of co-located network I/O intensive applications in a virtualized cloud. The experiments were taken on Xen platform. The study included the measurement of idle over running VMs, and the analysis in terms of throughput performance and resource sharing effectiveness. Authors showed that idle guest domains induce overhead which can impact the performance of CPU intensive applications in running domains. In the case of impact among identical neighbour applications, the default credit scheduler in Xen can approximately guarantee their fairness in CPU slicing, network bandwidth consumption, and the resulting throughput. However, strategically co-location of different applications in terms of workload rates and resource usage patterns can lead to considerable performance gain of about 40%. In particular, combining CPU-intensive and network-intensive applications results high aggregated throughput. Unfortunately, authors do not show how to utilize this strategy to help decision making in the cloud. Another important finding relates to the number of guest domains expending beyond some value, in which case the overhead raised by hosting multiple guest domains depletes the benefits of the best combination.

Casale et al. [CKK11] proposed performance models to predict the impact of consolidation on the storage (e.g., disks) I/O performance of virtualized applications. The models predict the throughput, response times, and mix of read/write requests of disk requests over a shared storage server. The approach measures certain quantities, such as the mean queue-length due to requests, to characterize read and write performance attributes. Measurements are obtained inside VMs with the block layer I/O tracing mechanism blktrace [11] tool, and at the storage server with the network

[11] http://linux.die.net/man/8/blktrace

protocol analyser tshark [12]. Both software tools are free and released under the GNU GPL license. Authors conduct the study by previously collecting and analysing data, in an isolated manner, for different workloads consolidation scenarios. Workloads are generated with FileBench [13] to emulate disk workloads of mail, web, and file server type applications. The results demonstrated to be fairly accurate in terms of read requests and throughput, with direct impact in response times of the applications in the VMs. Unfortunately, authors did not assess the effectiveness of their approach on consolidation of three or more VMs, and it is evident that write requests still to be hard to predict under consolidation.

Nathuji et al. [NKG10] proposed Q-Clouds, which utilizes online feedback to build a multi-input multi-output (MIMO) model to capture performance interferences among consolidated CPU-bound workloads. Then, it reacts to performance degradation by adjusting processor allocation for each application based on the required SLA. Through the hypervisor, it sets the cap mechanism on each VM to control the processing resources an application can use. It works by maintaining free resources to be lately used in the adjustment, without the need to determine the underlying sources of interference. Furthermore, the authors use Q-States to differentiate various levels of SLA so that they can dynamically provision underutilized resources, thereby improving system efficiency. Results show that performance interference is mitigated completely when resources are available, and the use of Q-States improves system utilization by up to 35%. Q-States vary from minimal performance that an application requires, up to hight levels of QoS defined by the customers willing to pay for them.

Zhu et al. [ZT12] proposed an interference model which predicts the impact of application performance due to interference from different levels of shared resources (e.g., on-chip, storage and network) under contention. Authors proposal consider time-variance in application workloads and use an influence matrix to estimate the additional resources required by an application to achieve desired QoS, defined in terms of execution time, as if it was executed alone. Then, leveraged by the proposed model to realize better workload placement decisions, authors introduce a consolidation algorithm to optimize the number of successfully executed applications in the cloud. The average prediction error is stated as being less than 8%. As a result, the number of successfully executed applications is optimized, outperforming static optimal approaches and a state-of-art approach such as Q-Clouds [NKG10], with negligible overhead. The approach implies a previous training period to profile the applications' behaviour when running co-located with others.

Rao et al. [RWGX13] developed a QoS provisioning framework that provides adaptive multi-objective resource allocation and service differentiation of multiple applications. It applies a hard limit over the CPU assigned to an application. However, the work does not consider effective utilization of shared virtualized resources, and it is limited to the regulation of a single resource.

Garg et al. [GTGB14] proposed a dynamic resource management strategy to handle scheduling of two types of applications, namely, compute intensive non-interactive jobs and transactional applications such as web server. The mechanism responds to changes in transactional workloads,

[12] http://www.wireshark.org/docs/man-pages/tshark.html
[13] http://www.solarisinternals.com/wiki/index.php/FileBench

and tries to maximize resource utilization whilst enforcing the SLA requirements and specifications. A web VM is scheduled based on the best-fit manner, not taking into account the resources consumed by VMs running dynamic jobs. In order to maximize the energy efficiency, the scheduler tries to explore free resources in nodes running web applications to schedule dynamic jobs. However if free resources are not sufficient so the task can be executed within its deadline, then the job is allocated in another node in the form of static job. Resources are transferred from dynamic jobs to web applications as demand changes. Periodically, the algorithm verifies the SLA accomplishment and realizes consolidation. The priority of scheduling is: (1) enforce SLAs; (2) schedule the jobs from batch job queue; and (3) consolidation. Despite authors have addressed directly the problem of heterogeneous workload management towards SLAs meeting, there is no consideration about performance degradation that occurs when co-located applications contend for resources. Moreover, authors do not consider the case in which consolidation of workloads may exceed the total available resources in the host, during runtime.

2.5 Thesis Scope and Positioning

This thesis focuses on optimization of IaaS environments, by creating and managing virtual clusters with enough resources to run users' applications under QoS constraints, and improving energy efficiency to run the workloads. The cloud provider is not aware of workloads that are executed in the VMs. Therefore, the cloud manager, which function is to manage the resources of the system, has to be application agnostic so as to handle arbitrary workloads. The goal of minimizing energy consumption while executing users' applications is achieved by dynamically consolidating VMs in fewer PMs, and switching idle PMs to sleep mode, as energy consumption is the major component of operating costs. By reducing energy consumption, CO_2 emissions to the environment are reduced as well. Because different types of applications, with diverse QoS requirements, can coexist and share physical resources, it is challenging to fulfil applications QoS requirements. The root of the problem is related with performance deviations caused by virtualization limitations in terms of performance isolation, and bursty oscillation in applications resources demand along runtime. To this end, this thesis investigates and proposes a mechanism to determine if applications QoS constraints are observed, as well as a proper scheduling algorithm that satisfies the QoS requirements of applications by assigning enough resources.

Two types of workloads are considered in this work, namely CPU- and network-bound workloads (e.g., HPC and Web), which have different QoS requirements. For CPU-bound tasks, performance deviation occurs in the form of slowdown in the execution of tasks. The slowdown varies along time, and according to the level of consolidation of VMs and properties of the workload. The approach proposed in this thesis consists of determining the tasks finish time and act according to detected slowdown in their execution. The performance deviation in the execution of CPU-bound tasks can be measured by state-of-the-art tools with some error, as in [DFB+12] that achieves an average error of 16%, due to changing in computing environment conditions. Regarding network-bound tasks, performance deviation manifests in the form of rate of requests not served per time

interval, and presents a bursty behaviour. In this case, monitoring of resource consumption gives hints about the deviation in the performance expected. A proposed scheduling algorithm leverages migration of VMs and the cap parameter from the Xen credit scheduler, to dynamically adjust CPU and bandwidth resources to VMs according to estimated performance deviation in the execution of applications. Unlike state-of-the-art cloud middle-wares such as OpenStack[14], Eucalyptus[15], and OpenNebula[16], this thesis proposes vertical and horizontal elasticity to deal with current issues in the cloud, in an integrated way. These are aspects distinguishing the work presented in this thesis compared to the related research.

There are a few related works reviewed in this chapter that are close to the proposed research direction. However, the work presented in this thesis is different in some aspects, and complementary in some others. Nathuji et al. [NKG10] focused on static VM consolidation using similar approach as it controls assigned CPU resources in order to mitigate performance interference among co-hosted tasks. In contrast, part of this thesis investigates the problem of horizontal and vertical dynamic consolidation, by constantly reading performance deviations along time in the execution of different types of applications, with diverse QoS constraints. Cumulatively, this thesis considers the power efficiency of IaaS environments, in order to optimize energy efficiency and reduce operational costs. Approaches to dynamic VM consolidation proposed by Zhu et al. [ZT12] model the performance deviation in terms of execution time, based on an influence matrix. The approach followed by the authors is similar, in the sense that they estimate the additional resources required by an application to achieve the desired QoS. This thesis goes further by providing a mechanism to take decisions upon provisioning of resources based on noisy slowdown data samples, and cumulatively considers diverse QoS constraints and energy efficiency optimization. Kim et al. [KEY12] considers both performance and power status to schedule tasks. However, their approach do not rely on real-time measures from performance deviation in applications execution time. The solution discussed in this thesis is power- and performance-aware in the construction and management of virtual clusters to run cloud users' applications with QoS constraints, and can dynamically handle QoS deviations more efficiently by timely responding to performance interferences.

2.6 Conclusions

The recently resurgence of virtualization relates to the possibility to deal with issues in the data center, such as hardware underutilization, variance in demanding of computing resources, high system administration costs, reliability and high-availability. Currently, virtualization technologies provide powerful mechanisms, such as VMs resizing, and migration, which administrators can utilize to adequately manage the computing environment. Cloud computing utilizes virtualization technologies in their systems as a manner to guarantee improved manageability, and provide fault isolation and dynamic resource scaling properties. Nowadays, cloud providers are faced with

[14] https://www.openstack.org
[15] https://www.eucalyptus.com
[16] http://opennebula.org

new challenges such as reduction of power consumption and guaranteeing SLAs, which is a key element to support and empower QoS in these environments. Consolidation of VMs is a typical methodology utilized to reduce power consumption, but can lead to unreasonable severe degradation of applications' performance. In fact, virtualization does not guarantee performance isolation between co-hosted VMs [HL13, KKB+07, PLM+13, GLKS11, KMHK12], meaning that applications performance can change due to the existence of others co-resident VMs sharing the limited underlying hardware resources. In this regarding, guarantee the expected QoS for applications, and reduce the energy consumption at the same time, is not trivial due to QoS deviations.

This chapter shows that management of computing resources can be done based on consolidation for improved resources utilization and reduction of operational costs, while maintaining similar levels of required performance by applications. One of the significant advancements that have facilitated the progress in managing compute servers is the implementation of models that predict the expected impact in performance of co-hosted applications. Other approaches try online measures of the slowdown experienced by co-hosted applications competing for resources. Based on such approaches, several scheduling algorithms, leveraging different techniques, have been proposed to mitigate the detected degradation in performance and commit the applications QoS requirements. This chapter presented various approaches to mitigate and compensate the loss of performance due to consolidation and resources sharing.

This chapter discussed advantages and limitations of virtualization technologies and positioned the current thesis in this field of research. The proposed research direction of this thesis is dynamic scheduling of tasks with disparate objectives, specifically considering energy efficiency and mitigation of performance deviations among co-hosted VMs, under performance constraints in IaaS clouds.

Chapter 3

Energy- and Failure-aware Data Center Resource Management

Cloud computing is progressively being adopted in different scenarios, such as business applications, social networking, scientific computation and data analysis experiments. In order to meet with the increasing needs of users, data centers are growing in size and in complexity. In parallel with this trend, data centers are faced with new challenges such as reduction of energy consumption and the ability to provide dependability at scale. In particular, the capacity to mask the effect of failures in physical servers is preponderant to provide SLA guarantees, which is a key element to support and empower QoS in these environments, as well as to reduce energy waste. This chapter presents an overview of power and energy concepts, identifies the sources of power consumption in a data center, and discusses fault tolerance schemes in computing systems.

3.1 Introduction

CLOUD computing apply extensive use of virtualization to implement the vision of IaaS, as described in Chapter 2. Despite the advantages from virtualization to manage resources in a flexible and efficient way, data centers supporting cloud environments continue to grow in size, meaning more nodes to manage and greater system management complexity as well. From the mid-1990s through today, designers of computing systems and the industry focused in the improvement of the system performance for less capital cost. In this regarding, the performance has been steadily growing driven by more efficient system design and increasing density of the components according to Moore's law [M+98]. Clusters greater than 10,000 processors [RCA11] have become routine in worldwide laboratories and supercomputer centers, and clusters with dozens and even hundreds of processors are now routine on university campuses [Cou10].

Unfortunately, the total power drawn by computing systems has not followed the constantly raising performance per watt ratio [Koo07]. The fact is that the average power use in different type of servers has been increasing every year, and the problem is even worse when one consider large-scale compute infrastructures, such as clusters and data centers. Koomey [Koo11] estimated that

energy consumption in data centers has risen by 56% from 2005 to 2010, and in 2010 accounted to be between 1.1% and 1.5% of the global electricity use. Today, the average data center consumes as much energy as 25,000 households [KFK08], and estimations point out it will continue to grow in the future. Taking into account that from a cloud provider's perspective the maximization of the profit is a high priority, the optimization of energy consumption plays a crucial role in the reduction of operational costs.

Tightly coupled with energy consumption, the term "green computing" was introduced with the aim of reducing carbon footprints. Data centers have a large and growing substantial CO_2 footprint. Kaplan study [KFK08] estimates that today's data centers result in more carbon emissions than both Argentina and the Netherlands. Raised by these environmental concerns, governments worldwide are approving laws to regulate the carbon footprint. For example, the European Union issued a voluntary Code of Conduct in 2007 prescribing energy efficiency best practices [C+10]. Another example of the efforts taken in this direction is The Green Grid, a global industry consortium focused on promoting energy efficiency for data centers and minimization of the environmental impact, has also forwarded recommendations for improved metrics, standards and technologies. The scope of sustainable computing includes not only computing nodes and their components (e.g., processors, memory, storage devices, etc.), but also other kind of related resources, such as auxiliary equipments for the computing facilities, water used for cooling, and even physical space that all these resources occupy.

In order to reduce data centers' inefficiencies (i.e., high electricity costs, huge carbon emissions, and server overload and low QoS), current solutions include: (i) more efficient cooling, to improve the PUE [BAHT11]; (ii) adoption of energy-efficient servers (e.g., voltage and frequency scaling); and (iii) consolidation of VMs on fewer servers (i.e., unneeded servers can be hibernated or used to accommodate more load). A recent work by Garraghan et al. [TXM14] has shown the impact of failures in a system in terms of energy costs, which can translate to almost 10% in the context of the total energy consumption for the entire data center. Furthermore, a study conducted by LANL estimated the mean time between failures (MTBF) to be 1.25 hours on a petaflop system, extrapolating from current system performance [Phi05]. As such, component failures become norms instead of exceptions. To this end, adoption of fault tolerance mechanisms is crucial in reducing energy consumption and providing dependable systems.

Cumulatively to tackling the problem of IaaS management to create performance-aware virtual clusters to fulfil users' applications QoS requirements (see Chapter 2), this thesis addresses the problem of optimizing dynamically the utilization of resources in order to reduce the energy consumption. Moreover, to comply with application QoS requirements, the management of resources should also consider the reliability of nodes to avoid energy waste and to provide high-available systems.

3.2 Energetic Characterization of a Data Center

This section presents an overview of power and energy concepts, analyses and identifies the sources of power consumption in a data center, and presents the mathematical model for power consumption by physical servers in modern data centers.

3.2.1 Power and Energy Models

In order to understand and determine the power being absorbed by any circuit element, it is essential to clarify associated terminology. Electric current is the rate of charge flow past a given point in an electric circuit, measured in coulombs/second. The standard metric unit for current is the ampere (A). A current of 1 A corresponds to 1 coulomb of charge passing through a cross section of a circuit wire every 1 second. The electric power associated with a complete electric circuit or a circuit component represents the rate at which work is done, and is measured in Watt (W), or Joules per second. In turn, energy is the total amount of work performed over a period of time, and is measured in Watt-hour (Wh) or Joule (J). For example, doing 100 J of work in one second (using 100 joules of energy), the power is 100 W. Mathematical expressions for power and energy are defined in Equation (3.1).

$$\begin{aligned} P &= \frac{W}{T} \\ E &= P \times T \end{aligned} \tag{3.1}$$

where P is the electrical power, T is the period of time, W is the total work performed during that period of time, and E is the energy. The power consumed by a PM can be decreased by lowering the CPU performance [VLWYH09], with consequent slowdown in execution of applications. In certain cases, the slowdown could be so extreme, that despite saving power the same amount of energy would be wasted, because the workload takes much longer to complete under consolidation [BZF10]. In turn, energy consumption can be decreased by using energy-efficient technologies and optimized hardware/software, and applying consolidation of the VMs on as few servers as possible. Idle servers can be switched to low power modes. If reduction of power consumption results in decreased costs of the infrastructure provisioning (e.g., costs of power generators, power distribution equipment, cooling system, etc.), reduction in energy consumption means decreasing the electricity bills.

3.2.2 Sources of Power Consumption

Lannoo et al. [L+13] conducted a study about energy consumption of data centers worldwide. The results are registered in Table 3.1. The authors considered the estimated average power use across three classes of servers, namely volume servers (less $25,000 per unit), mid-range servers (between $25,000 and $500,000 per unit) and high-end servers (more than $500,000 per unit), which were also considered previously in Koomey's study [Koo11]. According to the authors,

Table 3.1: Data centers worldwide power consumption in 2012 [L+13].

Server class	Volume	Mid–range	High–end
Power per server	222 W	607 W	8106 W
Storage		24% of total server power consumption	
Communication		15% of total server power consumption	
Infrastructure		PUE = 1.82	
Total energy	219 TWh	15 TWh	34 TWh

data centers worldwide are estimated to have consumed 268 TWh in 2012. The power consumed by storage and by communication was 24%, and 15%, of total server power consumption, respectively. A relevant fraction of the total energy consumed is devoted to the cooling systems and other overheads. The overall data center energy efficiency can be measured based on PUE, which is defined as the ratio of the total power consumed by a facility (data or switching center) with respect to the power consumed by IT equipment (e.g., servers, storage, routers, etc.). The metric introduced by the Green Grid focused on increasing the energy efficiency of data centers. The PUE is a factor greater or equal to 1, and according to US EPA report [B+08], it amounts to 1.9 on average for current data centers in the world. In other words, this means that the infrastructure overhead (e.g., cooling, power distribution, etc.) represents additional 0.9 W for every watt of power consumed in the computing equipment. In order to reduce the power consumption of cooling facilities, different solutions have been proposed [CK11, PGP10, KRA12], and some data centers, such as Google [RCA11], have even operating with a PUE of 1.12.

Regarding the IT equipment, the fraction of power consumption for storage and communication equipment represents 24% and 15% of the total IT equipment power consumption, respectively [L+13]. These values demonstrate that the most significant fraction of overall data centers power consumption is dictated by servers. According to Barroso et al. [BH09], the approximate distribution of peak power usage by hardware subsystem in one of Google's data centers is the one illustrated in Figure 3.1. As observed, the main part of power consumption in a server is dictated by the CPU, followed by the memory and losses due to the power supply's inefficiency. However, the power required by each component depends on the workload characteristics [ZL12].

There has been a continuous improvement of the power efficiency for the various computer components (i.e., CPU, RAM, disks, etc.) in a server, along the years. Most modern processors support Dynamic Voltage and Frequency Scaling (DVFS) [GMDC+13] technology, which dynamically changes the voltage and frequency of a CPU of a host according to its load to reduce the consumed dynamic power. The adoption of multi-core CPUs along with the increasing use of virtualization resulted in the growing amount of memory in servers, which has narrower dynamic power range [FWB07]. Additionally, even if a server is completely idle, it can still consume up to 70% of its peak power [MMP13, Len15]. Regarding power supplies which transform alternating current (AC) into direct current (DC), they achieve the highest efficiency at loads within the

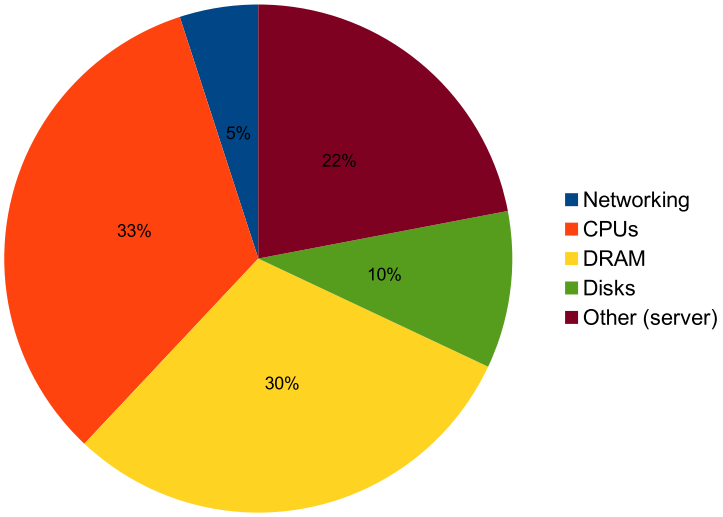


Figure 3.1: Approximate distribution of peak power usage by hardware subsystem in one of Google's data centers [BH09].

range of 40–60%, and almost maintaining it till 100% [BAHT07]. Considering that most data centers normally create a load of 11–50% [VAN08a], and that DVFS optimization is limited to CPUs, measures for reducing infrastructure energy consumption are needed to be taken. This thesis leverages VM consolidation to reduce the number of active servers, thus decreasing the energy consumption.

3.2.3 Modelling Power Consumption

Today's computer servers are equipped with power monitoring capabilities, which allow the monitoring of power consumption in real-time and collecting accurate statistics of the power usage. Based on collected data, it is possible to derive a model of dynamic power consumption for a particular system. Such a model should be able to predict the actual value of power consumption by a system, thus supporting the development of new policies for dynamic power management and understanding their impact in the whole system.

According to several studies and experiments (e.g., [RRK08, MDD10]), the power consumption by a server can be assumed, with the error below 10%, to grow linearly with the growth of the CPU utilization from the power corresponding to the idle state to the power consumed when the server is fully utilized. This relationship is observed in Figure 3.2.

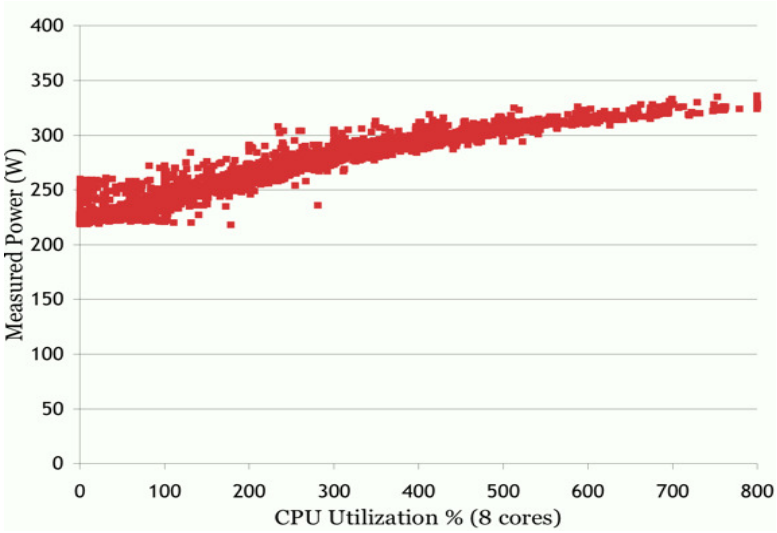


Figure 3.2: Power consumption versus CPU utilization for the 8-core Xeon server [RRK08].

This behaviour is captured by the Equation (3.2).

$$P(CPU) = p1 + p2 \times CPU\% \tag{3.2}$$

where P is the estimated power consumption, $p1$ is power consumption by an idle server ($CPU = 0$), $p2$ is the the additional power consumption from CPU utilisation, and CPU represents the current CPU utilization. Hence, the factor $p2$ is the dynamic power consumed by a PM, and is typically proportional to the overall system load. Fan et al. [FWB07] have derived more accurate non-linear relations, described in Equation (3.3).

$$P(CPU) = p1 + p2 \times (2 \times CPU\% - CPU\%^{\chi}) \tag{3.3}$$

where χ is a calibration parameter that minimizes the square error and has to be calibrated experimentally for each class of machines of interest. This thesis uses Equation (3.2) in analytical and simulation experiments, since refinements of Equation (3.3) have little practical utility [MMP13].

As stated by several studies and experiments (e.g., [FWB07, GHMP08, KGB13]), an active server with very low CPU utilization still consumes 50–70% of the power it would consume when fully utilized. This is due to the fact that power consumed by other system components (e.g., I/O, memory) is correlated with the CPU activity and have narrow dynamic power ranges. Even reducing the power consumption up to 70% by applying very-low CPU activity modes [BH07], other system components have narrower dynamic power ranges (e.g., less than 50% for DRAM, 25% for disk drives, and 15% for network switches). In this sense, resource consolidation and virtualization mechanisms allow development of several energy-aware resource allocation policies and scheduling algorithms to dynamically turn off idle machines to reduce the overall consumption.

3.3 Fault-tolerant Cloud Systems

Cloud systems focus on being scalable, high-available, cost efficient and easy access to resources on demand. As cloud systems are progressively being adopted in different scenarios, such as in scientific computing [VPB09], availability of a massive number of computers is required for performing large scale experiments. Nevertheless, some applications take a very long time to execute, even on today's fastest computer systems, which makes significant the probability of failure during execution, as well as the cost of such a failure in terms of energy and SLA violations. Moreover, cloud data centers become more prone to failure as infrastructures continue to grow in size and complexity, involving thousands or even millions of processing, storage, and networking elements. According to Schroeder and Gibson [SG07], three of the most difficult and growing problems affecting large-scale computing infrastructures is avoiding, coping and recovering from failures. In fact, in large-scale networked computing systems, computer component failures become norms instead of exceptions, making it more difficult for applications to make forward progress. The QoS perceived by consumers will be disastrously affected if measures to handle failure are not considered. Furthermore, failures result in a loss of energy because the computation is lost and needs to be repeated after recovery. To this end, it is evident that the success of ever growing warehouse-sized data centers being built to deal with increasing demand for computing resources will depend on the ability to provide reliability and availability at scale.

3.3.1 Relationship Between Faults, Errors and Failures

With the increasing popularity of cloud computing as an attractive alternative to classic management paradigm of distributed systems, it is of paramount importance to assure the correctness and continuous operation of the computing infrastructure, even in the presence of faulty components. According to Kondo et al. [KJIE10], *"a failure is an event in which the system fails to operate according to its specifications. A failure is observed as a deviation from the correct state of the system. (...) An error is part of the system state that may lead to a failure."*. The adjudged or hypothesized cause of an error is called a fault. In most cases, a fault first causes an error in the service state of a system's component, which in turn may lead to its subsequent service failure. Service failures, in this thesis abbreviated to failure, represents a transition from correct service to incorrect service, thus do not implementing the expected system function. In turn, service restoration stands for the transition from incorrect to correct service.

The root cause of the failures occurring in a system can be classified into human, environment, network, software, hardware, and unknown. In the context of this thesis, a failure is defined as any anomaly caused by hardware or software fault, unstable environment or intentional or mistaken actions carry out by the infrastructure administrator and that precludes computing infrastructure components to work. As reported in [SG07], Figure 3.3 shows failure statistics across all LANL systems. Failures in hardware represents the largest source of malfunction, accounting for more than 50% of all failures, whilst failures due to software places around 20% of all failures.

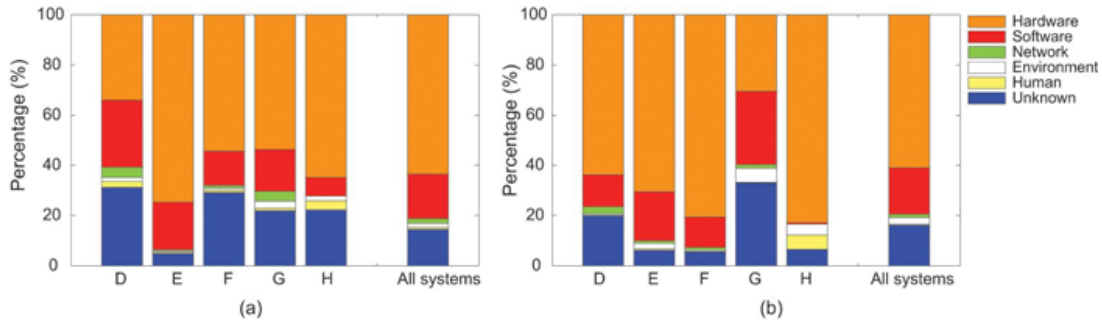


Figure 3.3: LANL systems statistics for [SG07]: a) The breakdown of failures by root cause. (b) The breakdown of total repair time spent on a system due to each root cause.

Moreover, according to the same study, an application running on all the nodes of a LANL system can be interrupted and forced into recovery more than two times per day. Considering that many scientific applications and experiments require a large number of nodes and weeks of computation to complete, failure and recovery are frequent events during an application's execution. Unfortunately, an important remark is that the failure rate of a system grows in proportion to the number of processor chips in the system, and technology changes over time has little impact in systems reliability. The consequences can be even more negative to currently executing applications if one consider that several of these nodes are frequently submitted to maintenance operations.

Reliability and availability [Sho02] are two measures applying how good the system is and how frequently it goes down. Reliability can be defined as the probability of no failure within a given operating period. In turn, availability is the probability that an item is up at any point in time. Both reliability and availability are used extensively as a measure of performance and to compare the effectiveness of various fault-tolerant methods. Availability of a system can be calculated, in a simple manner, based on uptime and downtime of the system. In this sense, availability is known to be a good metric to measure the beneficial effects of repair on a system. Rapid repair of failures in redundant systems greatly increases both the reliability and availability of such systems. Reliability can be measured by the MTBF, which in turn is estimated by the manufacturer of the component. In turn, mean time to repair (MTTR) is the time taken to repair a failed component, and it expresses responsiveness of the whole system. Availability is often measured by dividing the time the service is available by the total time, as expressed by Equation (3.4) [Kec02].

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{3.4}$$

Typically, the availability of a system is specified in nines notation. An availability of 99.9% (3-nines) refers to a downtime of 8.76 hours per year. The target availability in most telecommunication devices and many modern data centers is of 5-nines, that corresponds to 5 minutes per year. Both reliability and availability must be considered in order to measure systems performance. Both reliability and availability are attributes of dependability, which refers to the ability of a system to avoid service failures that are more frequent and more severe than is acceptable.

Over the last years, several techniques have been developed to improve systems availability. Avizienis et al. [ALRL04] grouped these techniques into four major categories, namely: (i) fault prevention, which aims at preventing the occurrence or introduction of faults in development process of systems (both in software and hardware); (ii) fault removal, which tries to reduce the number and severity of faults; (iii) fault tolerance, to avoid service failures in the presence of faults; and (iv) fault forecasting, to determine the future incidence of faults, and related consequences. Recently, fault tolerance and fault forecasting techniques have been used together in an effort to empower systems the ability and confidence in that it is able to deliver a service that can be trusted. Fault tolerance schemes, relying on fault forecasting techniques provided by state-of-the-art tools, as a way to provide system operation continuity and reduce energy waste represents one of the core themes of this thesis.

3.3.2 Achieving Fault-tolerant Systems

Dabrowski [Dab09] defined fault tolerance as *"the ability to ensure continuity of service in the presence of faults, or events that cause a system to operate erroneously"*. Indeed, fault tolerance mechanisms are aimed at failure avoidance, thus giving operational continuity to the system when one of the resources breaks down. Such technique is called failover. For that, fault tolerance mechanisms provide ways to carry out error detection and system recovery. According to Egwutuoha et al. [ELSC13], the major used fault tolerance techniques are redundancy, migration, failure semantics, failure masking, and recovery. A full revision about fault tolerance and fault tolerant systems is out of the scope of this thesis. Further information about this subject can be found in [KK10, ELSC13]. In short, redundancy, by hardware or software, and failure masking imply replication, which not only increases hardware/software costs and runtime costs, but also makes it more difficult to manage. In turn, failure semantics relies on the assumption of the designer to be able to predict failures and specify the most appropriated action to be taken when a failure scenario is detected. However, this assumption is surrealist due to complexity of today's computing environments such as cloud computing. Recovery implies recovering from an error so a system failure can be avoided. Nevertheless, not all states can be recovered, and applications may have to be restarted from scratch. Unlike reactive schemes, which means that faults are dealt after they take place, fault recovery commonly relies on checkpoint/restart mechanisms. Remus system [CLM+08], discussed in Chapter 2, is an example of reactive fault tolerance. However, the operation of checkpointing a process or a whole VM can incur significant overhead in a large-scale system. For example, the LANL study [Phi05] estimated that the checkpoint overhead prolongs a 100 hour job (in the absence of failure) by an additional 151 hours in petaflop systems. As a result, frequent periodic checkpointing based on current techniques often proves counter-effective.

In order to circumvent the problem of excessive overhead caused by traditional fault tolerance techniques, the trend on system dependability has recently shifted onto failure prediction and autonomic management technologies. Server failures can often be anticipated by detecting deteriorating health status, based on data collected using monitoring of fans, temperatures and disk error logs, etc. In this regarding, several research works addressed different techniques to estimate

the reliability of a server, by forecasting when the next failure will occur in that server. Therefore, proactive fault tolerance highly depends on the ability to predict the failure. Applications running in nodes that were incorrectly foresee as reliable will have to restart from scratch in case of node failures. However, in such cases the overhead of restarting a VM is negligible. Cavilla et al. [LCWS+09] demonstrated that it is possible to initiate multiple VM instances in less than 1 second, assuming that the VM images and data are stored in a network attached storage (NAS).

Despite failure prediction technique is crucial for resource management, it is out of the scope of this thesis. Some examples of work in this field can be found in [FX07a, FX07b, MN06, SLM10, YGK+10]. This thesis leverages the Song Fu's failure prediction tool [FX07a], so proposed scheduling algorithms can consider nodes' reliability and construct fault-tolerant and energy-efficient virtual clusters where users can deploy their jobs.

3.3.3 Proactive Fault Tolerance

In the last years, significant progress has been made in the field of failure prediction. Modern hardware devices are designed to support various features and sensors, that can monitor the degradation of an attribute over time and allow early failure detection [mbls10, Min06]. At the same time, several statistical- and machine learning-based prediction techniques have been presented with up to a considerable accuracy (e.g., [FX07a, MN06, YGK+10]).

Proactive fault tolerance relies on the assumption that node failures can be anticipated based on monitored health status. The technique can be combined with reactive fault tolerance schemes so as to adapt and reduce the checkpoint frequencies, and hence optimizing the trade-off between overhead cost and restart cost. Engelmann et al. [EVNS09] defined proactive fault tolerance (PFT) as *"(...) a concept that prevents compute node failures from impacting running parallel applications by preemptively migrating application parts away from nodes that are about to fail."*. Hence, PFT schemes do not need to implement redundancy nor replication. This property represents an advantage to achieve energy efficiency and optimize resources utilization in cloud data centers, because redundancy means extra power consumption. Moreover, relying in modern VM techniques, such as their ability of reconfiguration through VM migration, systems administrators are allowed to easily schedule plans of maintenance, reconfiguration, and upgrade in the data center infrastructure without interrupting any hosted services. In this sense, the construction of adaptive computing infrastructures with the support of VM technology, which adjust their configuration dynamically at runtime according to components' availability, represents an opportunity to efficiently utilize system resources for high-availability computing environments. This thesis tackles the problem of failures by applying proactive fault tolerance at granularity of VMs.

3.4 Energy- and Failure-aware Resource Scheduling

This section describes relevant work researching energy efficiency optimization and failure-aware resource scheduling topics.

3.4.1 Energy-aware Resource Scheduling

In recent years large volume of research has been done in the area of power- and energy-aware resource management at different levels. It is clear today that the next logical step is the development of data centers that are energy-efficient. Despite the considerable efforts towards building green facilities equipped with advanced cooling systems [HLM06, RCA11, KRA12] with a PUE near the unity, more attention has to be given to energy consumption of servers. Because average resource utilization in most data centers can be as low as 10% and the energy consumption of idle servers can be as much as 70% of peak power when are active [BH07], idle servers represent a major concern (even AC to DC conversion losses in power supplies of computer systems implies more loss in cooling). Figure 3.4 reproduces the range of power supply efficiency reported in [BAHT07].

A broad conclusion is that significant energy savings can be achieved in cloud computing through techniques such as VM consolidation, which explore the dynamic characteristic of workloads submitted to the cloud. The problem is seen as the traditional NP-hard bin-packing problem, in which several dimensions can be considered (e.g., CPU, RAM, network bandwidth, and disk). Greatly empowered by virtualization technologies that facilitate the running of several applications on a single physical resource concurrently, dynamic turning on and off servers method can be applied to reduce overall idle power draw and improving energy efficiency. In this regarding, some research simply use migration to aggregate the load on some servers and switch off idle ones, in order to improve energy efficiency [LO10]. Despite the impressive consumption peak, due to start of all the fans and the ignition of all the node components, by booting physical servers that have been switched off, an on/off algorithm is still more energy-efficient.

Meisner et al. [MGW09] proposed PowerNap which allows an entire server to rapidly transit between a high-performance active state to a near-zero-power idle state. In this sense, PowerNap energy-conservation approach aims at minimizing idle power and transition time, instead of requiring fine-grained power-performance states and complex load-proportional operation from server's components. Authors demonstrate that it is possible to substantially reduce the power consumption from 270 W in idle state to only 10.4 W in nap mode, in about 0.3 ms, for a typical blade server. Comparing PowerNap with DVFS approach, it shows to be more practical and efficient, in the sense that it requires system components to support only two operating modes, and provides greater energy efficiency and lower response time. According to PowerNap authors, the approach is suitable for real-world systems because the delay of transitions from and to the active state is less than the period of inactivity of most real-world systems, and the saved power is more than the power required to reinitialize the component from nap state.

Several studies based on PowerNap have followed, providing scheduling algorithms to dynamically adapt the assignment of resources to executing tasks, according to variation in workload demands. For example, Beloglazov et al. [BAB12] developed allocation heuristics provision computing resources to client applications (e.g., web applications) with the aim of improving energy efficiency of the data center, while guaranteeing required QoS. The strategy implies to consolidate

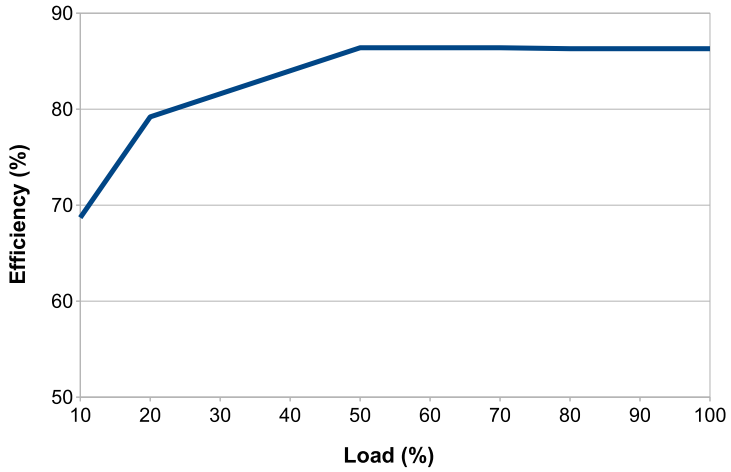


Figure 3.4: Power supply efficiency [BAHT07].

VMs in the fewest number of physical servers, leaving idle ones to go sleep. Authors modelled the power consumed by a server based on linear power equation, as expressed by Equation (3.2) in section 3.2.3. The strategy to allocate VMs is based on the Best-Fit Decreasing heuristic (BFD), and it works by iteratively picking up the VM currently utilizing more CPU and allocate it in the physical server leading to the least increase of power consumption. In order to improve the current VM allocation, and thus to optimize the energy efficiency and guarantee the QoS expectations, authors proposed three different strategies based on CPU utilization of a physical server. To this end, whenever the CPU utilization of a physical server falls outside the specified double-threshold interval, the strategy selects VMs that need to be migrated which then are placed, one by one, on the hosts using the modified BFD algorithm mentioned above. The results have shown that selecting the minimum number of VMs needed to migrate from a host performs better in terms of QoS constraints, and optimizes more energy efficiency compared to DVFS.

Lee and Zomaya [LZ12] conducted experiments over two proposed energy-conscious task consolidation heuristics. The main idea was to reduce the number of active resources so as to minimize the energy consumption for executing the tasks, without any performance degradation (i.e., without violating the deadline constraint). By dynamically consolidating tasks in fewer resources, idle physical servers can enter into power-saving mode or even be turned off. Proposed heuristics rely on the optimization of energy consumption derived from Equation (3.2), meaning that energy consumption is directly proportional to resource utilization.

Dhiman et al. [DMR09] contributed with vGreen to manage VMs scheduling across different physical servers with the objective of managing and improving the overall performance and system level energy savings. The proposal exploits internal characteristics of virtual machines (i.e., in terms of instructions per cycle, memory accesses) and their performance and power consumption. vGreen presents a client-server model: each physical server runs a client instance, which performs online characterization of the executing VMs, and sends update information to a central server. In

turn, the central server has the duty of schedule VMs across physical servers. The scheduling algorithm works by co-locating VMs with heterogeneous characteristics, by balancing three specific metrics that account for memory accesses per clock cycles, instructions per clock cycle, and CPU utilization for every VM. In the end, idle nodes in the system can be moved into low power states.

Rodero et al. [RJQ+10] studied the trade-offs of different VM configurations and their impact on performance and energy efficiency, for HPC applications. They proposed an autonomic strategy that combines online workload-aware provisioning and energy-aware resource configuration for clouds, in order to reduce the energy consumption in the data center, while ensuring QoS guarantees. It contributes with a decentralized online clustering to dynamically characterize and cluster the incoming job requests in terms of their system requirements and runtimes. Then, it explores the use of workload modelling techniques, and just-right dynamic provisioning, to reduce the energy consumption of the physical resources by powering down subsystems when they are not needed, and avoid over-provisioning of resources. Unlike [DMR09], this approach maps jobs with similar requirements to the same host system.

Ding et al. [DQLW15] focused in dynamic scheduling of VMs to achieve energy efficiency and satisfy deadline constraints. The considered scenario comprehends a cloud infrastructure comprising a set of heterogeneous PMs. Authors explore the idea that there exists optimal frequency for a PM to process certain VMs. The optimal frequency defines the optimal performance-power ratio to weight the heterogeneities of the PMs. The PM with highest optimal performance-power ratio is selected to process the VMs first, in order to save energy, unless it does not have enough computation resources. Authors proposed the EEVS scheduling algorithm, which can support DVFS technology to improve energy efficiency. The scheduling is divided into some equivalent periods, in each of which VMs are allocated to proper PMs and each active core operates on the optimal frequency. After each period, the cloud is reconfigured to consolidate the computation resources of the PMs to further reduce the energy consumption. Despite the interesting results in saving energy while deadline constraints of applications are met, the proposal ignores the performance and power penalties of status transitions of processor and VM migrations, limiting its deployment in real cloud environments.

Mastroianni et al. [MMP13] introduced ecoCloud, a self-organizing and adaptive approach for the consolidation of VMs on resources. Unlike most of the previous approaches, ecoCloud considers multiple dimensions (e.g., CPU and RAM) when making decisions on the assignment and migration of VMs to physical resources. The workload of each application modelled as a VM is dynamic, implying for example that CPU demand varies with time. The aim is to increase the level of utilization of servers by applying dynamic consolidation of workloads, with the twofold objective of increasing energy efficiency and maintaining the SLA stipulated with users. The decisions about configuration of VMs to physical servers mapping are, in a first stage, taken locally at each physical server which decides if accepts or rejects a VM according to differential between available and demanded resources, and then properly combined by the data center manager, in a high decision level. Because the problem is so complex to solve, and hardly scalable, assignment and migration of VMs are driven by ant algorithms [DGF+91] which consider local information

exclusively. Authors assess the performance of the approach extensively (i.e., analytical and experimentally), in order to cover a wider range of scenarios and without great assumptions.

Some approaches have considered the optimization of several objectives at the same time. In fact, power consumed by physical servers, thermal dissipation and maximization of resource usage are major concerns that have been subject of significant study. In this regarding, Xu and Fortes. [XF10] proposed a method to optimize conflicting objectives associated to initial virtual resource management. The size of a VM is represented as a d-dimensional vector in which each dimension corresponds to one type of the requested resources (e.g., CPU, memory and storage). Considered objectives are reduction of resource wastage, and operational power, and mitigation of individual hotspots by keeping temperature within a safe operating range. The problem of VM placement is formulated as a multi-objective combinatorial optimization problem that is solved as two-level control approach: (i) at first level, controllers implemented in every VM are responsible for determining the amount of resources needed by an application; in (ii) the second level, a global controller determines VM placement to physical resources. A grouping genetic algorithm [FD92] is utilized for combinatorial optimization (i.e., VMs to PMs mapping), and the solutions obtained are then evaluated using a fuzzy-logic system to optimize the several objectives. Later, [XF11] complemented the first work with a dynamic mapping of VMs to physical resources solution, which reassigns VMs to hosts due to changes of system conditions or VMs requirements. The objectives to achieve include elimination of thermal hotspots, minimization of total power consumption, and achieving desired application performance. First, the system analyses samples related to objectives within a temporal window for each PM, and if sample values are beyond pre-defined thresholds, then the system reconfigures VMs to PMs mapping by maximizing an utility function that combines all normalized objectives. The proposal was evaluated experimentally using a mix of types of workloads to emulate the variety and dynamics of data center workloads. Authors have shown that their approach significantly reduces unnecessary VM migrations by up to 80%, avoids unstable host selection, and improves the application performance by up to 30% and the efficiencies of power usage by up to 20%.

3.4.2 Failure-aware Resources Scheduling

In recent years, HPC systems have been shifting from expensive massively parallel architectures to virtual clusters deployed in cloud data centers [JD10, DEAS11, NBF+10, ZW11, KMB+11, ZW11, DSL+08, VPB09]. In this regarding, fault tolerance in such systems is a growing concern for long-running applications. Todays' research on system dependability has shifted onto proactive fault tolerance schemes. Thereby, the technique can be used in combination with dynamic scheduling algorithms to build failure-aware virtual clusters in cloud data centers. Aperiodic tasks whose arrival times are not known a priori must be scheduled based on such approaches [ZQQ11]. This section addresses solutions based only on virtual-machine-level fault tolerance, that is the theme followed in this thesis to provide construction of reliable and available virtual clusters. In this regarding, Nagarajan et al. [NMES07] was a pioneer in the promotion of automatic and transparent mechanism for proactive fault tolerance for arbitrary MPI applications. Authors exploit

Xen's live migration to migrate VMs from unhealthy nodes to healthy ones, employ the Intelligent Platform Management Interface (IPMI) [Min06] for hardware health monitoring and management of computing environment, and utilize Ganglia [MCC04], a scalable distributed monitoring system for HPC systems, to select the target node. The proposed approach implies that all nodes have an image of the overall system status. When the framework decides to migrate a VM, the selected node to where the VM will be migrated is the one having the lowest CPU load. The study conducted over several types of applications shows that the overhead due to migration of more than one single VM can be handled without much overhead, which is not affected by the problem size but mainly by the modification rate of pages during live migration and available network bandwidth. Moreover, unlike live migration, stop and copy migration results in constant time overhead. Most important, authors shows that such approach to handle failures is scalable, by testing it with different cluster sizes. Unfortunately, such approach considers only fault tolerance of the system, and does not consider other important objectives such as energy efficiency, when it prioritizes empty servers to host VMs migrated from about to fail nodes.

Song Fu has investigated for some years in the field of proactive fault tolerance, and has contributed with relevant methodologies in the correlation and prediction of failures [FX07a, FX07b]. Based on these contributions, the author has later proposed a failure-aware resource management approach in order to enhance systems availability and to achieve high performance [Fu10]. Basically, the author proposes two algorithm alternatives in the scheduling of deadline-driven jobs (a job is defined as a set of tasks), that consider both the capacity and reliability status of compute nodes, in the process of making selection decisions in provisioning of resources. The first scheduling alternative, Optimistic Best-Fit (OBFIT) algorithm, schedules jobs in nodes that are not predicted to fail before the jobs' deadline and cumulatively provide the minimum required resources to execute the jobs within their deadlines. The second alternative, called Pessimistic Best-Fit (PBFIT) algorithm, calculates the average available capacity level $C_{average}$ among the compute nodes that will not fail before job deadline, and from a set of PMs that will fail before user job deadline, it selects the PM with capacity C_p, such that $C_{average} + C_p$ results in the minimum available capacity to run a VM. Authors conduct experiments using failure traces from production systems and the NAS Parallel Benchmark programs on a real-world cluster system. The performance of the scheduling algorithms is assessed in terms of jobs and tasks successfully finished within their deadlines, and utilization of unreliable nodes, while changing the failure prediction accuracy of the tool [FX07a]. From experiences, authors show that system productivity can be enhanced by considering the reliability of nodes, as the job completion rate is increased by 17.6%, and the task completion rate reaches 91.7% with 83.6% utilization of relatively unreliable nodes. This approach assesses the risk of failure only at the time of (re)scheduling.

Butoi et al. [BSS14] proposed a distributed and autonomous fault-tolerant system for VMs running in a cloud computing environment. For that, authors run a local agent in each VM, which in turn uses fault tree analysis principles to process the events delivered by the hypervisor and to predict the reliability of the hosting VM. The fault-tolerant approach applies migration and replication of VMs. It is the duty of the local agent to assess and predict its VM health state and

to make decisions of when transferring the control to its VM replica or to start the live migration process. The approach addresses only the goal of fault tolerance. Health status is collected from local Xen log traces, considering the health status of VMs and not of PMs.

Polze et al. [PTS11] proposed a proactive fault tolerance scheme that considers health indicators at four system layers (i.e., hardware, hypervisor, guest operating system, and application layer). At each layer, the scheme comprises monitoring of system variables. Then, the outputs of all predictors at each layer are combined into one coherent evaluation of the current system state by applying ensemble learning method [Rok10]. Based on a global probabilistic reliability measurement, and in order to keep service-level response deadlines and availability, the proposed solution proactively moves VMs away from suspicious machines. Authors have demonstrated that the migration time increases directly with the amount of utilized memory, unlike the period of non-responsiveness to network I/O which remains practically constant. Nevertheless, authors did not present any further results about the approach, and did not identify the strategy to choose the PMs to which VMs would be migrated to.

Salami et al. [SSF+10] attempted to decrease unsuccessful job execution that may fail in the future, by not accepting them in order to not waste resources and increase system throughput. Authors have based their work on an extensive analysis about jobs execution in a LANL cluster, which considered failures in nodes and jobs characteristics, to apply job futurity prediction. Despite the promising results, the work is still in the very initial stage, and not acceptance of jobs for future scheduling may lead to false negatives, preventing some works to be accepted by the cloud.

Other approaches propose failure-aware scheduling algorithms without considering virtualization as the mechanism to leverage when processes need to migrate from about to fail nodes to healthy ones. However, the principles that support their functioning characteristics provide important information in the development of scheduling algorithms for virtualized environments. In this regarding, several failure-aware schedulers were proposed for different environments, such as clusters, grids, and desktop grids. For example, Oliner et al. [OSM+04] proposed a job spatial scheduling algorithm for allocation of parallel jobs on BlueGene/L, in the presence of failures. The algorithm idea is to choose nodes that have minimum failure probability during job execution. According to demonstrated results, the fault-aware scheduling can be effective even with modest prediction accuracy (around 10%). Also Li and Lan [LL06] exploited failure prediction, by proposing an adaptive fault management mechanism called FT-Pro, with the aim of reducing the application execution time in presence of failures. A cost-based evaluation algorithm is proposed to adaptively select different preventive actions based on the accuracy of failure prediction. To this end, the scheme dynamically chooses the preventive action, which can be proactive process migration or reactive checkpointing, during the execution of parallel applications. The results show that combining different fault-tolerant techniques, the application execution time for long-running applications can be reduced by 13–30%, despite failures.

Some authors leveraged PFT in the scheduling of workflows. According to Arabnejad and Barbosa [AB14], a workflow can be defined as a set of interconnected steps that must be executed to accomplish some work. Usually, a workflow is described by a Directed Acyclic Graph (DAG),

where the vertices are the tasks and the edges are the temporal relations between the tasks. Because workflows constitute a common model for describing a wide range of applications in distributed systems, several projects have designed workflow management systems to manage the execution of applications in distributed systems. In this regarding, Yusuf et al. [YSP09] started by proposing recovery-aware service components (i.e., self-contained autonomous processes such as server processes running in a VM), which combines prediction with reactive and PFT techniques to improve the reliability of workflow applications. It works by pre-emptively rebooting components to probabilistically safer states whenever probability of failure goes beyond a threshold. Later in [YSP11], the same authors extended the work to support MapReduce workflow applications. Both works targeted the grid. In [YS13], Yusuf and Schmidt have analysed their last contribution in a cloud computing environment, and also evaluated the case of combinational logic architectural patterns which includes, for example, massive real-time streaming, big data processing or scientific workflows. The results allowed authors to conclude that their fault tolerance approach improves systems reliability compared to using only prediction-based fault tolerance or reactive fault tolerance.

3.5 Thesis Scope and Positioning

This thesis investigates dynamic multi-objective scheduling of VMs under QoS constraints, where physical resources are subject to failure. Objectives include reduction of cloud providers operational costs, by decreasing the energy consumption by the physical resources that are rented to cloud costumers. Also, the scheduling algorithms target reliability of computing environments by applying proactive fault tolerance against failure events, i.e., any anomaly, or intentional or mistaken actions carry out by the infrastructure administrator, and that preclude computing infrastructure components to work.

Targeting the management of IaaS cloud environments, this work dynamically manages the construction of energy-efficient and fault-tolerant virtual clusters, to which users workloads are scheduled and executed. Cloud computing providers are under great pressure to reduce operational costs and carbon emissions to the environment. It is therefore extremely important to act on the level of energy efficiency, in order to optimize the usage of resources and minimize the impact of failures in energy consumption [TXM14]. However, achieving energy efficiency and accomplishing applications QoS constraints simultaneously is a complex task.

The approach proposed in this thesis follows horizontal and vertical scaling of VMs as a mean to optimize power efficiency and thus reduce the operational costs, and considers nodes' reliability in the provisioning of resources. By continuously monitoring physical servers CPU and reliability, based on state-of-the-art failure prediction tools [Fu10], it is possible to detect energy optimizing opportunities and to obtain the predicted occurrence time of the next failure. Power- and failure-aware scheduling algorithms use monitored data to construct and manage power-efficient and dependable virtual clusters, while maintaining service-level performance. In order to extensively evaluate the performance of the approach proposed in this thesis, sets of simulations and

experiments in real cloud testbed are carried out, with randomly generated workloads and with workloads that follow data centers tracelogs of reference. These are aspects distinguishing the work presented in this thesis compared with the related research.

As presented in Section 3.4, recent research efforts have provided related work to the proposed in this thesis. However, the work presented in this thesis is different in crucial aspects, by trying to focus in solving some important and specific issues nowadays cloud providers are facing. For example, while several works (e.g., [BAB12, LZ12, DMR09, RJQ+10, ZZA10, KBB09, MMP13, XF10]) proposed solutions and alternatives aiming at reducing operational costs through energy efficiency optimization, they lack the importance of fault tolerance in large-scale computing systems. On the other hand, several researchers addressed the problem of reliability and availability (e.g., [NMES07, BSS14, PTS11]), by proposing techniques missing the objective of energy efficiency, or providing solutions based on checkpointing and replication (e.g., [JCC+10, PPF09, PGMAdM11]). The solution discussed in this thesis is power- and failure-aware in the construction and management of virtual clusters to run cloud users' tasks, and yet is able to fulfil applications QoS requirements.

3.6 Conclusions

Nowadays, cloud computing providers are experiencing great pressure to reduce operational costs and carbon emissions to the environment, so maximization of the profit can be a reality. However, guaranteeing service level agreements of users' applications and optimizing energy efficiency at the same time is hard to achieve because both objectives can become difficult to conciliate. Additionally, recent studies show that failures are a source of inefficiencies that increment energy consumption in large-scale data centers [TXM14]. In fact, as cloud data center infrastructures continue to grow in size and complexity, failures become norms instead of exceptions, making it increasingly difficult for applications to make forward progress [SG07]. The QoS perceived by consumers can be seriously affected, and hence the success of petascale computing will depend on the ability to provide reliability and availability at scale. In this regarding, mechanisms to reduce energy waste and to provide fault tolerance are essential to achieve energy efficiency and dependability at scale in future cloud data centers.

In the context of this thesis, a failure is defined as any anomaly caused by hardware or software fault, unstable environment or intentional or mistaken actions carry out by the infrastructure administrator and that precludes computing infrastructure components to work. Failures impose high implications on the applications deployed in VMs which justifies an increasing need to address users' reliability and availability concerns. Hsu and Feng [HF05] showed in a recent work that the existing reliability of large high-performance clusters is constrained by a MTBF varying from 6.5 h to 40 h, depending on the maturity of the installation. Also a study conducted by LANL [Phi05] estimates from current system performance that the MTBF would be 1.25 h on a petaflop system. In spite of these numbers, system designers and administrators are becoming more concerned about failure occurrences as well as their impact on system performance and

operational costs.

Current research has proposed isolated solutions to improve energy efficiency. Some of those mechanisms intend to optimize energy efficiency by providing scheduling algorithms that consolidate VMs according to specific objectives such as temperature-, performance-, and power-aware management [XF11]. Other approaches, such as the mechanism proposed by Nagarajan et al. [NMES07], offer to strengthen the dependability of a system by leveraging failure prediction and migration to move VMs between about to fail and spare physical servers. However, these mechanisms tend to neglect power efficiency, in particular those which implement replication and checkpoint, and negatively affect energy efficiency at the expenses of system dependability. In turn, Song Fu [Fu10] implemented a failure prediction tool, based on a study over failures occurring in a LANL cluster system, which later came to be used in the building of two failure- and power-aware scheduling algorithms. The work consisted in one of the first approaches to integrate power- and failure-aware construction of virtual environments for execution of cloud users' applications.

This chapter has concluded with a discussion of the scope and positioning of the current thesis in the context of energy efficiency and fault tolerance schemes, and discussed advantages and limitations on their use. The proposed research direction of this thesis is multi-objective dynamic scheduling of tasks, specifically considering energy efficiency, and fault tolerance, under performance constraints in IaaS clouds.

Chapter 4

Building Energy-efficient and High-available Virtual Environments in the Cloud

This chapter presents a multi-objective approach to energy-efficient and high-available dynamic and reconfigurable distributed VM infrastructure for networked computing systems. In this approach, the resource manager efficiently utilizes system resources for high-availability computing and energy efficiency optimization to reduce operational costs and carbon footprints in the environment. The resource manager combines an energy optimizing mechanism to detect and mitigate energy inefficiencies, and executes power- and failure-aware decision making algorithms, that leverage virtualization tools, to dynamically construct and readjust virtual clusters to enable the execution of users' jobs with a high level of SLA accomplishment. The higher efficiency of the proposed resource manager is shown by extensive simulations and real platform evaluation, by injecting random synthetic jobs and jobs using the latest version of the Google cloud tracelogs.

4.1 Introduction

As mentioned in Chapter 3, a crucial requirement for effective consolidation is the ability to efficiently utilize system resources for high-availability computing and energy efficiency optimization to reduce operational costs and carbon footprints in the environment. This chapter presents a mechanism for detection of power inefficient PMs and scheduling heuristics for the problem of energy-efficient and fault tolerance dynamic and reconfigurable distributed VM infrastructure for networked computing systems. The proposed algorithms consolidate VMs when needed and consider nodes reliability, in order to minimize power consumption and increase dependability of virtual environments. The computing of resources assignment to VMs is dynamically updated along time, and QoS constraints are taken into account.

Cloud computing implements the concept of utility computing, where users pay for access to computing resources according to their expected usage. However, not all cloud users request

the contracted computing power for the entire period of the contract. For example, the study elaborated by Meisner et al. [MGW09] showed that server utilization is below 30% in typical data center deployments, which causes energy waste. Garg et al. [GTGB14] refers that currently cloud data center providers prefer static VM allocation over dynamic, which leads to inefficient utilization of resources and exacerbates the problem of operating costs. Also failures contribute to energy waste. Justifying this fact, Townend et al. [TXM14] have analysed the relationship between failures and energy waste in a large-scale cloud environment, concluding that more than 20% of energy waste results from failure events that preclude tasks to execute normally. In large-scale networked computing systems running a variety of different jobs, component failures and consequent node unavailability are a characteristic. A LANL study [Phi05] reported that a node's MTBF is 1.25 h in a petaflop system.

The target of this study is an IaaS, where the provider infrastructure is composed of a cloud manager and a cloud scheduler modules, to construct and dynamically manage energy-efficient virtual clusters to execute sets of independent tasks in which computing resources are affected by failures. The cloud provider is application-agnostic, i.e. unaware of applications and workloads characteristics served by the VMs of the virtual clusters. This study measures the service performance based on the success rate of job completion by the deadline. The SLA, or contract, is defined for each job by specifying a deadline guaranteed by the system. Since energy costs are becoming a more significant factor in the operational costs of modern businesses, it is natural to consider energy consumption into the service cost equation and provide high service-level performance [BAB12], rather than ensuring absolute performance at any cost. The cloud manager module optimizes the infrastructure by reacting to energy inefficiencies and continuously monitoring power consumption. The proposed approach is adequate to implement scalable and elastic service platforms on demand because of its dynamic characteristic. At the same time, the same module improves availability of virtual clusters and maximizes the rate of completed jobs by applying proactive failure tolerance technique. It can be used as well to adapt and to reduce stop times in maintenance operations, in which case the administrators should only specify the hosts and their maintenance operation times. The cloud scheduler module is responsible for solving the problem of dynamically mapping tasks, running on virtual machines, to hosts in a power- and failure-aware manner. These virtual-to-physical resource-mapping decisions consider the performance status, power efficiency, and reliability levels of the computing nodes. The optimization decision includes selecting the CPU capacity necessary to accomplish the tasks within their specified deadlines, considering the predicted MTBF of each node and the migration overhead of the VMs. The optimization decision includes selecting the CPU capacity necessary to accomplish the tasks within their respective deadlines, considering the predicted reliability of each host and the migration overhead of the VMs. Specifically, the proposed approach to dynamic reconfiguration of virtual clusters consists in solving the 4 sub-problems:

1. Deciding if a host is considered power-inefficient, so that all VMs should be migrated away from it, and the host should be switched to a low-power mode sate.

2. Deciding if a host is considered reliable enough, so that VMs running longer than predicted MTBF should be migrated away from it to other active or reactivated hosts to avoid violating the QoS requirements.

3. Selecting the most appropriate host, in terms of power efficiency and reliability, to which VMs can be scheduled.

4. Provisioning VMs with the right amount of resources to avoid waste but preventing scarcity as well.

Contrasting with the studies discussed in Chapter 3, the strategy proposed in this thesis enables the autonomous rescheduling of the tasks currently running to complete the submitted users' jobs in an energy-efficient and reliable manner. By ensuring that tasks are completed by their deadlines, the SLA imposed on each job is being satisfied. The proposed modules are evaluated by extensive simulations and real platform evaluation. The workloads utilized in tests were generated randomly and based on realistic workloads from recent Google cloud tracelogs [Goo11]. The main contributions of this study are:

1. The development of power- and failure-aware cloud scheduling algorithms that implement vertical and horizontal platform elasticity.

2. The development of a dynamic scheduling strategy to provide power- and failure-aware virtual clusters that reactively detects energy optimizing opportunities and perform consolidation to reduce energy consumption while maintaining service-level performance.

3. An extensive evaluation of the dynamic strategy and the scheduling algorithms with randomly generated workloads and with workloads that follow the latest version of the Google cloud tracelogs.

4. A dynamic configuration of the CPU portion assigned to each VM that reduces the consumed energy and maintains the service-level performance.

The rest of the Chapter is organized as follows: Section 4.2 presents the architecture of the proposed cloud computing manager to enforce construction of power- and failure-aware virtual clusters to run users' jobs, formulates the problem, and introduces the techniques to support the solution of the problem. Section 4.3 describes the proposed scheduling algorithms for VM placement and provisioning, as well as state-of-the-art algorithms for comparison purposes. Section 4.4 introduces the metrics used to evaluate the performance of the proposal, the characteristics of the workloads and failure occurrences, and describes the simulation scenario and testbed for experiments. The results and discussion is done in Section 4.5. Section 4.6 is the final conclusion of the chapter.

4.2 An Approach to Energy- and Failure-aware Virtual Clusters

The target system is a private cloud IaaS environment, deployed over a common data center consisting of h homogeneous physical nodes. In a typical scenario, the cloud users utilize and share resources from cloud provider, in a pay-as-you-go manner.

4.2.1 System Overview

The architecture of the considered cloud management system, information flow, and relative key components involved in power efficiency estimation and failure-aware scheduling of applications is illustrated in Figure 4.1. The cloud infrastructure is composed of h physical hosts, such that $M = \{m_1, \ldots, m_h\}$, where M is the vector representing the PMs. The servers in the cloud are homogeneous, which means that PMs have the same CPU capacity C, memory capacity R, LAN network bandwidth N_1 and Internet network bandwidth N_2, access to a shared storage space S for storing the disk images of the VMs, and predicted time in the future for the occurrence of failures F_i, which depends on the PM under consideration, such that $m_i = \{C, R, N_1, N_2, S, F_i\}$. The high-speed LAN link N_1 is used for resource management purposes, such as virtual machine migration, signalling and control data, etc. The link N_2 is utilized by cloud users to access cloud services. Stressing N_1 manifests itself in the overhead of VM migration. An independent system estimates and provides the amplitude of the migration overhead to the cloud management system, hence influencing the decisions made by the scheduling algorithm. The blocks "Power efficiency" and "Reliability" in Figure 4.1 continually monitor the PMs status, and support the cloud manager in making decisions on resource configuration. As indicated in the beginning of this chapter, in the context of this study a failure is considered as any anomaly caused by hardware or software fault, an unstable environment, or intentional or mistaken actions carried out by the infrastructure administrator that precludes the computing infrastructure components from working properly.

In a typical usage scenario, users ask for resources to the cloud manager to then execute their jobs. The cloud manager is responsible for evaluating the capacity required, and for deciding whether to accept or reject service requests according to the servers' ability to guarantee QoS requirements. By ensuring that tasks are completed by their deadlines, the SLA imposed on each job is satisfied. Otherwise, a penalty to the service provider will be applied. A job $j = (T_j, d_j)$ aggregates a set of n independent CPU-bound tasks, $t_q \in T_j, q \in \{1, \ldots, n\}$, and is constrained by a deadline, d_j. Once tasks are independent, the job deadline becomes the deadline of the longest task. The task workloads are expressed in Mflops. For the sake of simplicity, a VM encapsulates one single task, and is the unit of migration in the system. Each VM runs on top of one PM at a time. Is the responsibility of the cloud manager to create and to manage the VMs to execute the tasks, which constitute the user's virtual cluster execution environment. A job is scheduled only if a VM can be assigned to each task of the job, and when scheduled, the deadline is activated.

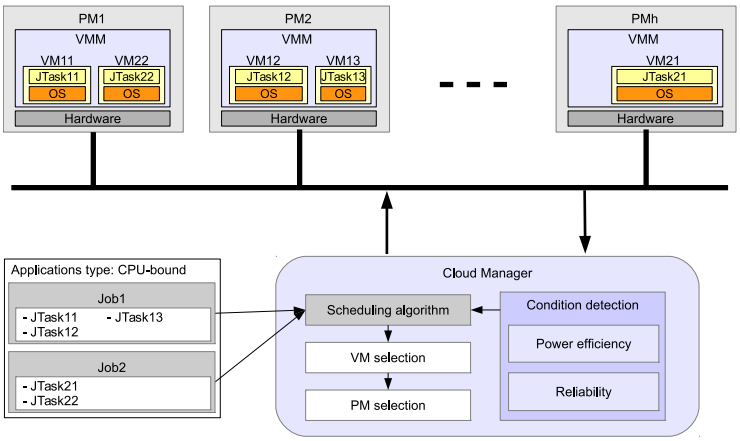


Figure 4.1: Private cloud management architecture.

4.2.2 Problem Formulation

Although computing nodes in data centers are composed of several components, such as memory, network interface cards, and storage disks, the power consumption is primarily determined by the CPU [BGDG+10, BAB12]. In this regarding, the proposed cloud management system considers only the CPU power consumption in the energy model. As discussed in Chapter 3 power consumption by servers can be estimated based on the linear power model described by Equation (3.2). Considering this relationship between power consumption and CPU utilization, Equation (4.1) defines the power efficiency of a PM i at a specific sample time, which increases monotonically with the workload, reaching 1 at 100% CPU usage [XF10, XF11]. It reflects how much useful work is produced for a given power consumption. The $p1$ and $p2$ factors, which represent the power consumption when the PM is in idle mode and the additional power consumption from CPU utilization, respectively, are used to normalize the efficiency. A task may be executed on a single host, or may migrate between hosts more than once because of node failures or power inefficiencies. A failure prediction mechanism [Fu10] specifies the MTBF for each machine.

$$E_{P_i} = \frac{CPU_i\%}{p1 + p2 \times CPU_i\%} \times (p1 + p2) \tag{4.1}$$

By efficiently consolidating the VMs, the average power efficiency of the execution of all jobs can be optimized, for all active physical nodes u, at all sample times f, as described by Equation (4.2). As stated in Chapter 2, VMs competing for resources suffer from performance degradation, which causes slowdown of running applications and ultimately a deviation between expected and delivered QoS. In this study, the estimated performance degradation is accommodated when selecting the PMs and assigning a CPU percentage to each task.

$$\overline{E_P} = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f}, \forall u \leq h \tag{4.2}$$

4.2.3 Exploiting the PM States

Several studies (e.g., [BGDG+10, BAB12]) showed that an idle state host can consume as much as 70% of the power consumed when it is running at maximum CPU utilization. A reduction of power consumption can be effectively achieved by switching idle PMs to sleep mode in response to instantaneous loads. Modern real-world servers allow near-zero overhead transitions to the sleep mode [NPI+08] consuming low power. For example, Meisner et al. [MGW09] showed that a typical blade server consuming 450 W at peak can transit rapidly in a fraction of a millisecond to a near-zero-power idle state, consuming approximately 10.4 W. Thus, the power consumption can be effectively reduced in modern real-world systems with low-latency transitions to the sleep mode. Other techniques such as DVFS are less efficient, since they work only over the CPU system component, resulting in lower gains of energy [BAB12].

4.2.4 Exploiting Virtualization Features

As indicated in Chapter 2, this study utilizes virtual machine stop and copy migration technique either for energy optimization or failure tolerance. Fu [Fu10] compared the performance between stop and copy migration to the live migration, in terms of time taken to conclude the process of migration. The results demonstrated that the migration of VMs with the stop and copy method requires 12–14 seconds, in contrast to the 14–25 seconds required for a live migration, on average. Live migrations introduce imperceptible downtime of the migrating VM [CFH+05], but take longer to complete because the copy of memory pages to destination node is a function of the frequency of the writing to the memory pages, and dirty memory pages must be re-copied. In contrast, stop and copy migration time is mostly constrained by the network bandwidth used to completely transfer the stopped VM, to then be resumed in the destination PM [HG09]. Thus, stop and copy migrations accomplish the migration process in a shorter interval of time than live migrations [NMES07, Fu10], which is of paramount importance when a failure is predicted to occur in a node.

As for resource allocation to VMs, this study defines a cap value [CGV07] to control the upper limit of CPU time that can be consumed by a VM. As introduced in Chapter 2, this feature is available for example on Xen credit scheduler. Several studies (e.g. [ZZA10, NESS09]) showed that dynamic provisioning of CPU at runtime reduces power consumption, and can serve as an effective control actuator for power budgeting. Moreover, by dynamically adjusting the CPU cap using the Xen credit scheduler, the fraction of CPU resources assigned to each VM can be updated to explore the available CPU fraction in the physical node until the sum of the caps of all co-located VM instances is 100% [SSGW11].

4.2.5 Power Efficiency Detection Mechanism

The management of virtual-to-physical resource mappings and VM migration to improve the efficiencies of resource usage and power consumption in data centers has been subject of considerable amount of investigation (e.g. [XF11, BGDG+10, BAB12, MGW09]). One of the most important

mechanisms in the chain of energy optimization process consists in the effective detection of power inefficiencies.

This section proposes a sliding-window condition detection mechanism to detect energy optimizing opportunities. The output of this mechanism is then used by the cloud manager module to interact with the scheduling algorithms to efficiently manage the infrastructure in a power- and failure-aware manner. This mechanism can be tuned regarding the window length containing a certain number of samples for analysis, the threshold value, specifying the minimum fraction of CPU utilization in the physical server below which an event is triggered, and the required number of detected events within the sliding-window so the cloud manager initiates a consolidation action. The CPU utilization samples are taken at 1 minute intervals. This work henceforth represents the specified CPU utilization threshold value as τ, and by γ the number of events detected within a sliding-window. Moreover, the energy efficiency improving mechanism applies stabilization measures, by imposing that the next sliding-window samples are ignored for a PM to which all executing VMs have migrated at less than γ samples. This policy aims at avoiding cloud infrastructure instability and consequent lower rate of completion jobs due to additional, and often unsuitable, number of migration processes occurring during optimization of the current VMs allocation.

The mechanism does not define any maximum CPU utilization threshold value since it is assumed that running tasks do not meet resources scarcity during execution, as cloud manager only schedules tasks if there exists the minimum necessary amount of required resources to complete the jobs by their deadlines.

4.2.6 The Cloud Manager

The cloud manager module is responsible for the optimization of the infrastructure by reacting to energy inefficiencies and failures that are predicted to occur. For that, the cloud manager continually obtains the virtual and physical machine statuses, such as the nodes' reliability, power consumption, and execution progress of the tasks. Then, the cloud manager makes decisions based on the information collected. Energy efficiency is improved through VM consolidation mechanism, by transferring VMs from lower loaded PMs to other PMs, and by putting the first PMs in sleep mode. A limit to CPU resources assigned to VMs is defined by setting the cap value, which increases the level of consolidation of VMs, and decreases the number of active PMs (by augmenting the load rate of the active PMs). Forecasting the time at which the next failure is going to occur in a certain PM is determined using the tool described in [Fu10, FX07a]. In case a PM fails unpredictably or fails before its running VMs conclude the migration process, those VMs will be re-initiated in spare PMs.

Algorithm 1 describes the cloud manager algorithm. In line 8, all the VMs running in PMs that are predicted to failure are migrated to spare ones, ζ minutes before the failure occurs in the preservation of the work completed so far. In line 9, the VMs that were running on PMs that failed, before migrating their VMs, are added to the job list to be initiated again. Tasks that have not yet been scheduled in the last scheduling instant and tasks from the new jobs that, meanwhile, have arrived, are added to the job list in line 10. In line 6, if there are no other tasks to schedule, the

consolidation mechanism is executed to improve the power efficiency. This mechanism is based on a single threshold detection method to identify underutilized PMs. All the tasks running in such PMs are added to the list to be migrated to more power-efficient PMs. However, if there are other tasks to schedule, the consolidation is postponed because those tasks will change the load on the PMs. In line 12, a scheduler algorithm is called to map the list of available tasks to the PMs, regardless of whether they are new tasks or tasks to be migrated. Only tasks that can start immediately are mapped, whilst the remaining ones stay on the unscheduled list for later scheduling. Such an approach enables the scheduling algorithm to execute faster and is more adequate to the dynamic behaviour of a cloud. If a valid map results from the scheduling algorithm, it is applied in line 14. The map includes the initiating of new tasks, as well as the migration of running tasks. If more than one migration occurs from a source or to a destination PM, they occur sequentially in time. After the map is applied, the idle machines are collected in line 16 and are set to sleep mode in line 17. Then, the algorithm waits for an event in line 19. The events that may occur are a PM failure, a PM failure estimate, a VM consolidation event, completion of a task, or the arrival of a new job. Every time one of these events takes place, the cloud manager module starts a new scheduling iteration. The provisioning of CPU fraction assigned to a VM may be changed only if it is rescheduled from a migration operation or re-spawning of a task.

Algorithm 1 Cloud manager algorithm.

```
1:  function CLOUDMANAGER(pmList, cloudUsers)
2:      Event e ← NULL
3:      while true do
4:          jobList ← NULL
5:          if e == ConsolidationEvent then
6:              jobList.add(pmList.getTasksFromPMsPowerInefficient(γ, τ))
7:          else
8:              jobList.add(pmList.getTasksFromPMsAboutToFail(ζ))
9:              jobList.add(pmList.getTasksFromPMsFailed())
10:             jobList.add(cloudUsers.getUnscheduledJobs())
11:         end if
12:         map ← schedAlg(pmList, jobList.getTaskList())
13:         if map ≠ NULL then
14:             executeMapping(map)              ▷ Executes the VMs to PMs mapping
15:             pmIdleList ← NULL, map ← NULL
16:             pmIdleList.add(pmList.getIdlePMs())
17:             pmSetSleepMode(pmIdleList)
18:         end if
19:         e ← WaitForEvent()
20:     end while
21: end function
```

4.3 Power- and Failure-aware Scheduling Algorithms

The cloud manager invokes the algorithms to carry out task (re)scheduling, in a power- and failure-aware manner. Because the problem of mapping the VMs to the PMs is NP-complete, the proposed algorithms are heuristic. The submitted jobs and their tasks are heterogeneous in terms of resource requirements and deadlines. Each task is defined by three properties, namely, workload size, maximum execution speed, and deadline. These properties are used by the algorithms to prioritize the list of tasks. Likewise Stillwell et al. [SVC12], this study assumes that a task t only requires the maximum amount of resources $max\ r(t)$ necessary to execute at the maximum speed, a value defined by the user. Based on the deadline defined by the task, Equation (4.3) defines the minimum resources $r(t)$, in Mflops/s, assigned to a task t that are necessary to complete its remaining workload $W(t,\mu)$ in the time from time $\mu + m_{ei}$ to the task deadline d_t. The parameters μ and m_{ei} represent the current time and the overhead required for a VM to migrate from node e to node i, respectively. The parameter m_{ei} is 0 if the task is scheduled for the first time or re-initiated. As an example, lets consider the matrix multiplication algorithm that requires $2n^3$ flops for matrices of size (n,n) to compute. If the deadline d_t available to execute the task is 120 s, the required resource is $2n^3/120$ flops/s.

$$min\ r(t) \geq \frac{W(t,\mu)}{d_t - \mu - m_{ei}} \tag{4.3}$$

Based on the task deadline d_t and the minimum time necessary for task completion, which is achieved when maximum resources are assigned to it, the slack time of task t is determined as shown in Equation (4.4). Tasks in the list for which the slack time is a negative value are cancelled, and the corresponding jobs are considered incomplete, incurring an SLA violation. In order to satisfy the required SLA for each job, the algorithm assigns to the task a CPU fraction between $min\ r(t)$ and $max\ r(t)$.

$$slack\_time = (d_t - \mu) - \frac{W(t,\mu)}{max\, r(t)} \tag{4.4}$$

The next two sections describe the two algorithms proposed in this study. In both algorithms, the CPU resource assigned to a VM is re-evaluated whenever a migration takes place, which means that the CPU requirement of the source PM may be different from that of the destination PM. Such updating action over the CPU assigned to a VM only occurs when migrations are performed, thus allowing horizontal and vertical scaling of the VMs with respect to the CPU allocation. Aiming at creating a set of virtual clusters, one for each user, to execute the tasks associated with the incoming jobs $\{j_1, \ldots, j_k\}$, the cloud manager invokes the algorithm to select the VM to schedule and determine a destination PM.

4.3.1 Power- and Failure-aware Relaxed Time Execution Algorithm

The first algorithm proposed in this work is POwer- and Failure-Aware Relaxed time Execution, here henceforth called POFARE. The algorithm is described in Algorithm 2. In order to improve

the overall power efficiency and availability of the system, the optimization of the current VMs to PMs mapping is carried out in two steps: (i) selection of VMs that can be migrated; (ii) placement of VMs in the PMs capable of improving the power efficiency and reliability of the system. The pseudo-code is described in Algorithm 2.

VM Selection

The schedule of VMs follows a very well determined priority according to the reason why they are in the scheduling list, and according to their properties. In this regarding, the algorithm starts by creating a list of prioritized groups of tasks $\{t_1, \ldots, t_{n \times k}\}$. The priorities are, from the highest to the lowest priority (1) proactive failure tolerance; (2) re-initiating of failed tasks; and (3) scheduling of new tasks. Power-optimizing scheduling has the least priority, being performed when there are no other types of tasks. Then, the tasks in each group are sorted in ascending order according to the slack time of the tasks (see Equation (4.4)). These steps correspond to lines 3–5 of the algorithm POFARE.

Algorithm 2 POwer- and Failure-Aware Relaxed time Execution (POFARE).

```
1:  function POFARE(pmList, jobList)
2:      mappingList ← NULL
3:      jobList.taskList.removeTasksHavingNegativeSlackTime()
4:      jobList.taskList.groupTasksByReason()
5:      jobList.taskList.sortTasksInGroupsByIncreasingSlackTime()
6:      for all task ∈ jobList.taskList do
7:          lRweight ← NULL                          ▷ lowest reliability weight
8:          hEpower ← NULL                           ▷ highest power efficiency
9:          pmSelected ← NULL
10:         for all pm ∈ pmList do
11:             if pm.hasResources(task) and pm.hasReliability(task) then
12:                 rWt ← pm.getReliabilityWeight(task)
13:                 ePw ← pm.getPowerEfficiency(task)
14:                 if ePw > hEpower or (ePw = hEpower and rWt < lRweight) then
15:                     lRweight ← rWt
16:                     hEpower ← ePw
17:                     pmSelected ← pm
18:                 end if
19:             end if
20:         end for
21:         if pmSelected ≠ NULL then
22:             mappingList ← {task, pmSelected}
23:             pmSelected.updateResourcesAvailable(task.minResources)
24:         end if
25:     end for
26:     return mappingList                           ▷ VMs (tasks) to PMs mapping
27: end function
```

VM Placement

After a VM is selected to be scheduled from the prioritized list, in line 6 of Algorithm 2, all PMs are evaluated with respect to task resource requirements, PM power efficiency, and reliability. By using Equation (4.5), in line 11 the algorithm checks if the node i can supply the minimum resources required by task t. The parameter $r_i(t_{i_k})$ expresses the resource assigned to task t_{i_k}, while the set of tasks $\{t_{i_1}, t_{i_2}, \ldots, t_{i_n}\}$ is running on node i. A machine i is considered a valid candidate if it provides enough capacity C_i to account with the minimum resource demand of t, which is $min\ r(t)$, plus the overall resources required by all tasks that run on that machine.

$$r_i(t) + \sum_{k=1}^{n} r_i(t_{i_k}) \leq C_i \tag{4.5}$$

After checking as valid the first condition in line 11, the algorithm assesses if node i accomplishes with the second condition expressed by Equation (4.6). This equation indicates that task t can be (re)scheduled or migrated from node e to a non-reliable physical node i at time $\mu + m_{ei}$, which is predicted to fail at instant δ_i, if cumulatively (1) it provides the resources $r_i(t)$ required by task t during the $\delta_i - \mu$ time interval and (2) task t will have to migrate to another, currently unknown physical node where it cannot require more resources than those needed to enable execution at maximum speed, $max\ r(t)$.

$$\begin{aligned} r_i(t) \times (\delta_i - \mu - m_{ei}) + (max\ r(t)) \times (d_t - \delta_i - m_{ei}) \geq W(t,\mu), \\ \text{if } ((\delta_i - \mu - m_{ei}) \leq \frac{W(t,\mu)}{max\ r(t)} \wedge r_i(t) \in [min\ r(t), max\ r(t)]) \end{aligned} \tag{4.6}$$

The above equation implies that the node i is predicted to fail before the task deadline d_t, thus a virtual machine migration will happen. However, if the task t starts at time $\mu + m_{ei}$ at node i, with a predicted instant of failure δ_i later than the task deadline d_t, then the task can be allocated in that node without additional migrations. In this case, the Equation (4.7) is evaluated to calculate the amount of resources necessary to complete the task within the deadline. If node i does not satisfy Equation (4.7), it is discarded to receive the current task.

$$\begin{aligned} r_i(t) \times (d_t - \mu - m_{ei}) \geq W(t,\mu) \\ \text{if } ((\delta_i - \mu - m_{ei}) > \frac{W(t,\mu)}{min\ r(t)} \wedge r_i(t) \in [min\ r(t), max\ r(t)]) \end{aligned} \tag{4.7}$$

After selecting a set of PMs that complies with Equations (4.5)–(4.7), in line 21 the algorithm selects the node that best improves the power efficiency of the machine (i.e., Equation (4.1)) and that can provide the required resources. If there is more than one node providing the same power efficiency E_P, in line 14 the algorithm selects the one with the highest reliability. According to Equation (4.8), which expresses the reliability weight for a given node i, the node with lowest R_i

is the most reliable. The policy applied by Equation (4.8) forces the algorithm to allocate tasks that have a low slack time in nodes that have the necessary resources to avoid migrations.

$$R_i = \frac{1}{2^{\delta_i - d_t - m_{ei}}} \tag{4.8}$$

In the end, the algorithm sets the cap parameter in the Xen credit scheduler for fine-grained CPU assignment, allocating the strictly necessary amount of resources to complete the tasks within their deadlines. Nevertheless, the algorithm dynamically adjusts the resources assigned to VMs to execute tasks in work-conserving mode, by exploiting remaining free CPU capacity of the node. For that, the remaining free CPU capacity is distributed equally among all running VMs until the entire CPU capacity of the node is consumed or the maximum capacity of all VMs is reached. Thus, tasks tend to run at the maximum possible speed. As stated in the biginning of the chapter, replacement of VMs is achieved through stop and copy migration technique. Considering n tasks, and a cloud data center composed of h nodes, the complexity of the algorithm is $O(n \times h)$.

4.3.2 Power- and Failure-aware Minimum Time Execution Algorithm

The second algorithm proposed in this work is POwer- and Failure-Aware Minimum time Execution algorithm, here henceforth called POFAME. POFARE applies a hard limit to reserve the minimum required resources to complete the task by its deadline, thereby increasing the task's completion time. Unlike POFARE algorithm, POFAME tries to reserve the maximum amount of resources needed to execute the task (in line 23 of Algorithm 2), only limited by the available PM resources and the maximum amount of resources required by the task, $max\ r(t)$. As for POFARE, the complexity of POFAME is given by the product of the number of tasks, n, to be scheduled at a given time and the total number of machines, h. For a single iteration, the complexity of both algorithms is $O(n \times h)$.

4.3.3 State-of-the-art Scheduling Algorithms

In order to better assess the performance of the proposed algorithms, three other algorithms where implemented to compare their performance with that of the algorithms proposed in this study. The first algorithm, Common Best-Fit and here henceforth called CBFIT, is a best-fit type algorithm that is based on the MBFD algorithm presented by Beloglazov et al. [BAB12]. The CBFIT strategy selects, from all available PMs, the PM that has the minimum capacity necessary to run a task to optimize energy consumption. Because MBFD does not consider the reliability of the nodes, CBFIT is a simplified version to show the impact of the proactive fault-tolerant migrations. The other algorithms, OBFIT and PBFIT, are failure- and power-aware scheduling algorithms proposed by Song Fu [Fu10], and properly discussed in Section 3.4.2.

4.4 Performance Evaluation

This section introduces the concepts in the basis of the performance evaluation of the proposed framework (i.e., energy optimizing mechanism and scheduling algorithms) to construct and manage energy-efficient and high-available virtual clusters. The concepts introduced are the metrics used to evaluate the performance of the energy optimizing mechanism and algorithms, the characteristics of the workloads and failure occurrences. The simulation scenario, and a experimental testbed are also described.

4.4.1 Simulation Setup

As the targeted system is an IaaS, a cloud computing infrastructure as described in Figure 4.1 is essential to evaluate the proposed scheduling algorithms for a large set of workloads and failure rates. Because it is found to not be feasible or practical to guarantee repeatable conditions of experiments on a real infrastructure [BAB12], simulations were used to evaluate the algorithms.

The simulated cloud computing infrastructure comprised 50 homogeneous physical nodes, each of which having the CPU capacity of 800 Mflops/s. The performance characteristics of nodes in terms of CPU can be determined using Linpack [DL11]. The power consumed by the fully loaded physical nodes is 230 W, which, according to the SPECpower benchmark [SPE14] for the first quarter of 2014, represents an approximate value for modern servers. The parameters $p1$ and $p2$ for Equations (3.2)(4.1) were set up to 70% and 30% of full power consumption, respectively. Regarding memory usage, a VM requires a RAM size of 256, 512, or 1024 MB, randomly selected. The migration overhead of a VM depends on the memory size and the network bandwidth, which, in this experiment, was set to 1 Gigabit/s [BAB12, HG09]. The scheduling algorithms have no knowledge of when jobs arrive.

The deadline of a task can be either before or after the predicted occurrence time of node's next failure. If a failure is predicted to occur before the deadline, the task migrates $\zeta = 3$ minutes before the node's predicted failure time. To forecasts when the next failures will occur, this work leverages the failure predictor tool proposed by Fu [Fu10, FX07a], which can predict failure occurrences with an average accuracy of 76.5%. To detect energy optimizing opportunities, the mechanism was applied based on a sliding window with size of 5 CPU usage samples, with a CPU usage threshold τ varying within $\{55, 60, 65, 70, 75, 80, 85\}$% and number of events detected γ (i.e., needed to trigger a consolidation action) within the window varying in $\{1, 2, 3\}$.

In a first stage, this work evaluates the performance of the energy optimizing mechanism, by evolving τ and γ parameters, and then it measures the impact of failure prediction accuracy on the performance of the scheduling algorithms, as well as the average task length to MTBF ratio, when the prediction accuracy is 75%.

4.4.2 Performance Metrics

In order to evaluate and compare the performance of the algorithms, three metrics were defined, and following explained.

The Completion Rate of Users' Jobs

The completion rate of users' jobs E_J is expressed as Equation (4.9). It measures the completion rate of jobs, which is calculated as the ratio of the number of jobs completed by their deadline, J_C, to the number of submitted jobs, J_S. The value of metric E_J falls in the interval [0, 1], and it is the SLA metric. In this sense, the difference between E_J and 1, multiplied by 100, gives the percentage of SLA violations.

$$E_J = \frac{J_C}{J_S} \tag{4.9}$$

The Energy Efficiency

The energy efficiency, E_M, is show in Equation (4.10). It measures the amount of energy consumed, in Joules, to produce useful work. The useful work is calculated as the number of Mflops associated with successfully completed jobs only, J_C. In turn, the energy consumption is determined by multiplying the average power consumption of the computing infrastructure (i.e., for all active physical nodes u at all sample times f) by the number of sample times f, multiplied by 60 (because the samples are obtained each minute). Finally, E_M is calculated by dividing the sum of the workloads from all tasks of successfully completed jobs by the overall energy consumption. The energy efficiency metric is represented as Mflops/Joule.

$$E_M = \frac{\sum_j (\theta_j \times \sum_{t=1}^{n} W(t,0))}{\frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} P_i}{u}}{f} \times f \times 60}, \quad \theta_j = \begin{cases} 1, & \text{if job } j \text{ completed} \\ 0, & \text{otherwise} \end{cases} \tag{4.10}$$

The Working Efficiency

The working efficiency, E_W, is introduced in Equation (4.11). This metric is used to determine the quantity of useful work performed (i.e., the completion rate of users' jobs) by the consumed power. It is calculated by multiplying E_J, the completion rate of jobs, by the average power efficiency based on Equation (4.1), for all active physical nodes $i \in [1,u]$ at all sample times f.

$$E_W = \frac{\sum_{s=1}^{f} \frac{\sum_{i=1}^{u} E_{P_i}}{u}}{f} \times E_J, \; \forall u \leq h \tag{4.11}$$

The Equations (4.10)–(4.11) represent metrics that capture the amount of useful work performed from different perspectives. While the first metric quantifies the number of useful Mflops

by the consumed energy, the second measures the quantity of useful work (i.e., completion rate of users' jobs) performed with the consumed power. The best algorithm should be the algorithm that maximizes both, enabling the processing of more Mflops with a lower amount of energy and maximizing the job completion rate while keeping high levels of power efficiency.

4.4.3 Workload Characteristics

The performance of the scheduling algorithms differs according to the workload considered. To extensively analyse how well the algorithms perform, two different kind of workloads were injected in the cloud system. This section describes the characteristics of those workloads.

Random Synthetic Workloads

A set of synthetic workloads were created based on the Poisson distribution, which was already used in the literature [BGJ06] to describe the characteristics of jobs in distributed systems. The average job inter-arrival time was set to 10 min, and each job was composed of an average of 10 tasks. Additionally, the average task length to MTBF ratio varied on a logarithmic scale of $\{0.01, 0.1, 1, 10\}$. For example, a ratio of 1 means that the average task length equals the MTBF. The average CPU utilization per task was set to 20% of the node capacity. The injection of this type of workload into the simulator enabled an analysis of the impact of the average task length to the MTBF ratio on the performance of the scheduling algorithm. Considering these workload characteristics, a set of 100 synthetic jobs was created. The average amount of CPU resources required by a VM running a random workload was set to 160 Mflops/s (i.e., 20% of a node's capacity), meaning that the maximum number of VMs per node is 5, on average. The task deadlines are rounded up to 10% more than their minimum necessary execution time, and the deadline of the longest task defines the job's deadline.

Workloads Based on Google Cloud Tracelogs

Recently, there has been some notable efforts to provide a comprehensive in-depth statistical analysis of the characteristics of workload diversity within a large-scale production cloud [MGTX14, TXM14, RTG+12]. In the basis of these studies is the second version of the Google Cloud tracelog [Goo11], which contains over 12,000 servers, 25 million tasks and 930 users over the period of 29 days. These studies yielded significant data on the characteristics of submitted workloads and the management of cluster machines. Their contribution enable further work on important issues, such as resource optimization, energy efficiency improvements, and failure correlation. The analysis performed clarified that approximately 75% of jobs only run one task and most of the jobs have less than 28 tasks that determine the overall system throughput. The average length of a job is 3 minutes, and the majority of jobs run in less than 15 minutes, despite the fact that there are a small number of jobs that run longer than 300 minutes. Moreover, task length follows a lognormal distribution, with most of the tasks requiring a short amount of time. This same distribution applies to CPU usage, which varies from near 0% to approximately 25%, indicating that

a high proportion of tasks consume resources at lower rates. For the same reason, a lognormal distribution can be applied to describe the number of tasks per job. Depending on the cluster or day observed, job inter-arrival times follow distributions such as lognormal, gamma, Weibull, or even exponential, with a mean time of 4 seconds. Most of the tasks use less than 2.5% of the node's RAM.

Based on these studies, a set of 3614 synthetic jobs was created to simulate cloud users' jobs. Each task requires a RAM size of 256, 512, or 1024 MB, selected randomly. The jobs comprised a total of 10357 tasks. Each task deadline was rounded up to 10% more than its minimum necessary execution time, and the task with the longest deadline defines the job deadline.

4.4.4 Failures and Unavailability Properties

Dependability and operational costs due to energy consumption highly depend on the impact of failures within a system. Therefore, the design of scheduling algorithms for constructing energy-efficient and high-available virtual clusters onto data centers requires a good understanding of failure characteristics. This section describes the characteristics of the failures injected in the simulator to evaluate the performance of the scheduling algorithms.

For each physical node, the MTBF was programmed according to a Weibull distribution, with a shape parameter of 0.8. It has been shown [SG07] to well approximate the time between failures for individual nodes, as well as for the entire system. Failed nodes stay unavailable (i.e., MTTR) during a period modelled by a lognormal distribution, with a mean time set to 20 min, varying up to 150 min. Failure tolerance relies on proactive VM stop and copy migration, rather than checkpointing or live migration. The predicted occurrence time of failure is earlier than the actual occurrence time. When a node fails, the tasks running on it are restarted in a different set of nodes, from scratch, if there is enough time to execute those tasks before their deadlines. The (re)initiating of VMs is assumed to impose negligible overhead [LCWS+09].

4.4.5 Testbed Setup

Simulation represents a useful approach for analysis of the performance of proposed algorithms, since it provides a controlled environment which guarantees repeatable conditions for a set of experiments. However, it is important to analyse the performance of the proposed algorithms in a real platform. Because state-of-the-art cloud system middle-wares do not implement vertical scaling of a task, it was developed a prototype of the Cloud Manager (see Algorithm 1).

The experimental testbed consisted of 24 nodes (cores) from an IBM cluster with the following characteristics: each physical server was equipped with an Intel Xeon CPU E5504 composed of 8 cores working at 2.00 GHz, supporting virtualization extensions and deploying 24 GB of RAM. The Xen 4.1 virtualization software was installed in the Ubuntu Server 12.04 operating system. Reliability of nodes follows the characteristics reported in Section 4.4.4. The cloud manager is aware of nodes' status by collecting heartbeat messages [ZMSM10]. A failure is considered when a node stops sending heartbeat messages.

To emulate the tasks workload following the Google cloud tracelogs, as explained in Section 4.4.3, the stress software (available on Ubuntu Linux) was used. A set of 151 jobs with a total of 389 tasks was created, which ran for approximately one hour to complete an instance of the experiment.

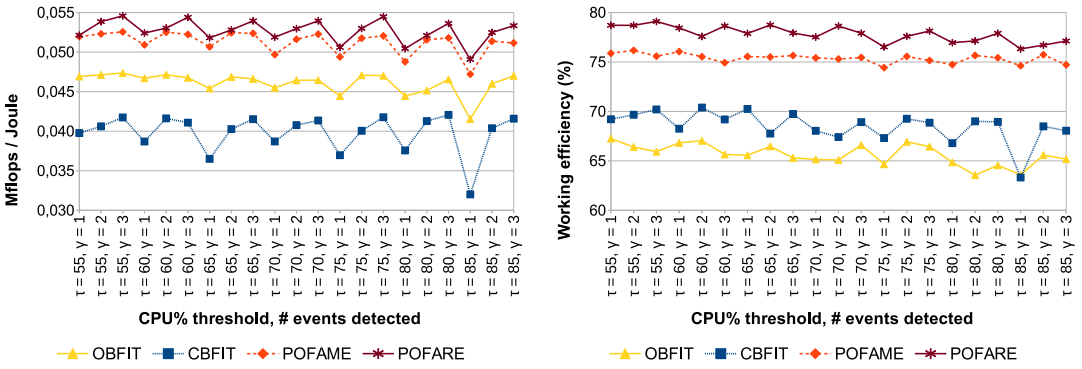
4.5 Simulation Results and Analysis

The energy optimizing mechanism and scheduling algorithms are extensively assessed throughout several set of simulations and experiments in real cloud testbed. First, a set of random workloads is injected in the simulator to investigate how well the algorithms perform for tasks with different average length to MTBF ratios. Then, Google-based workloads are used to assess the algorithms performance with and without the energy optimizing mechanism that dynamically re-adapts the schedule to improve power efficiency. The section ends with the reporting of preliminary results from the adoption of the proposed algorithms in a real platform.

4.5.1 Energy Optimizing Mechanism

Figure 4.2 depicts the results provided by the energy optimizing mechanism, for diverse combination of values of the parameters CPU utilization threshold τ, and number of events detected γ. This mechanism was introduced in Section 4.2.5. The study utilized the set of workloads following the last version of Google cloud tracelogs. The MTBF, MTTR, and accuracy of the failure prediction tool, was set to 100 minutes, 20 minutes, and 75%, respectively. The schedule of VMs is done by utilizing the algorithms in study. The results for PBFIT were omitted due to its low performance. Figures 4.2(a)–4.2(b) show that the values $\tau = 55\%$ and $\gamma = 3$ result in the best combination to optimize energy efficiency and working efficiency metrics, thus producing more work for less energy.

Considering the above results, the energy optimizing mechanism was tuned with $\tau = 55\%$ and $\gamma = 3$, to produce an optimal ratio of the amount of work performed to the consumed energy. Next, the performance improvements on energy efficiency and work efficiency were assessed driven by the utilization of the energy optimizing mechanism. The results are presented in Figure 4.3, for MTBF, MTTR, and average failure prediction accuracy fixed at 100 minutes, 20 minutes, and 75%, respectively. Figures 4.3(a)–4.3(b) evidence the improved energy efficiency and working efficiency results due to the use of consolidation (i.e., use of energy optimizing mechanism) compared to the case of non use of consolidation. In particular, POFARE algorithm outperforms all the other alternatives, and it improves energy efficiency and working efficiency in 9.7% and 15.6%, respectively, with the use of dynamic consolidation mechanism. Figure 4.3(c) depicts the number of migrations and respawns occurred in the system, with and without consolidation of VMs. Despite the consolidation mechanism led to an increase in the number of migrations, the number of migrations remains low. In the specific case of POFARE (i.e., the algorithm that performs better regarding metrics considered), the rate of migrations increased from 0.32% to 2.26%, regarding the total number of VMs executed in the system. Figure 4.3(d) shows the completion rate of user's



(a) Energy efficiency.

(b) Working efficiency.

Figure 4.2: Tuning energy optimizing mechanism parameters τ and γ, with workloads based on Google cloud tracelogs. The configurations are: (i) MTBF = 100 minutes; MTTR = 20 minutes; (iii) average failure prediction accuracy = 75%.

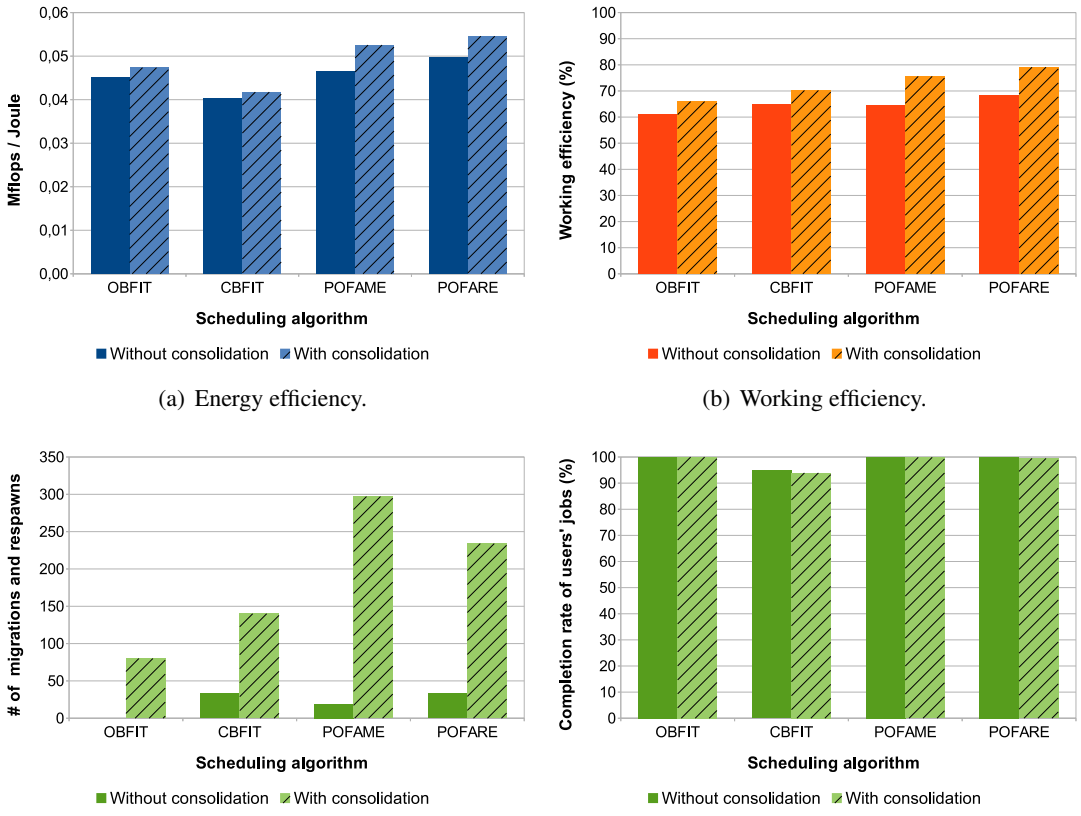
jobs. By applying the consolidation mechanism, POFARE algorithm suffered very small negative impact of less than 0.48% in the completion rate of users' jobs.

4.5.2 Simulation with Random Synthetic Workloads

The results for set of simulations using the synthetic workload characteristics are shown in Figure 4.4. The MTBF was fixed at 200 minutes, and the average task length to MTBF ratio was varied within the set $\{0.01, 0.1, 1, 10\}$. The average prediction accuracy for the failure predictor tool was set at 75%, and the results were obtained without dynamic consolidation. Since CBFIT algorithm does not consider node failures, it is used as the baseline for performance comparison.

Figure 4.4(a) shows the job completion rate as a function of the average task length to MTBF ratio. The graphic shows that as the average task length increases compared with the MTBF, the completion rate of users' jobs decreases. For the values of 0.01 and 0.1, task lengths are small compared with the MTBF and OBFIT, POFAME and POFARE perform similarly, completing almost 100% of the jobs. CBFIT only performs well for small ratio values, because the task lengths are small compared with the MTBF and failures do not affect its performance. As the ratio increases, CBFIT spends most of the time re-initiating tasks. OBFIT performance decreases as well as the ratio augments, since it selects nodes that do not fail until the task deadline. As the ratio increases, less number of adequate nodes can be selected to place the tasks. In contrast, as the ratio augments, PBFIT tends to complete more jobs when dealing with unreliable PMs, as it only schedules tasks to unreliable PMs. For the ratio of 0.1, POFARE and POFAME complete 99% of the jobs. Although it is a high rate of success, it is not 100% because of the additional overhead caused by the migration of the VMs to tolerate node failures. The migration technique shows significant improvements for the ratios of 1 and 10 compared with the other algorithms.

Figure 4.4(b) shows the average power efficiency along simulation, a measure of consolidation level. As the task sizes increase, the nodes run at a higher percentage of CPU utilization. POFARE



(a) Energy efficiency.

(b) Working efficiency.

(c) Number of migrations.

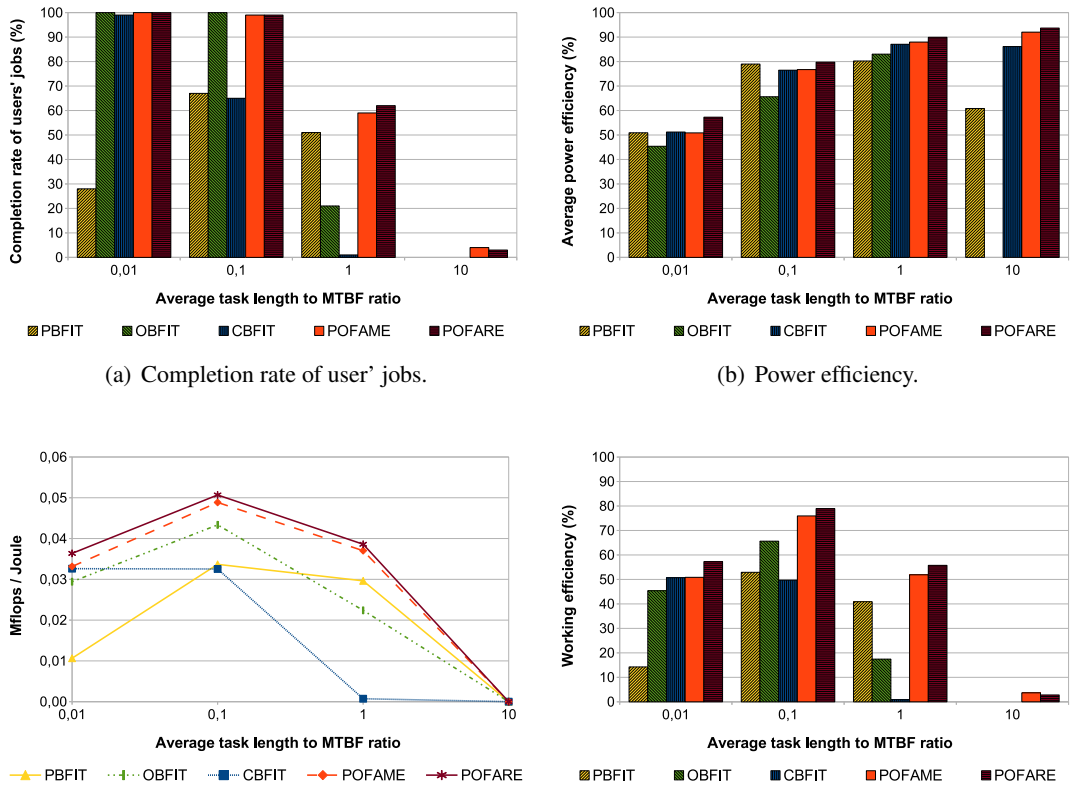
(d) Completion rate of user' jobs.

Figure 4.3: Impact of consolidation of VMs in energy consumption, with workloads based on Google cloud tracelogs. The configurations are: (i) MTBF = 100 minutes; MTTR = 20 minutes; (iii) average failure prediction accuracy = 75%.

is the best algorithm in all cases, going above 90% for the ratios 1 and 10. For the ratio of 10, the power efficiency is 0 for OBFIT because it schedules tasks in reliable PMs only. The other algorithms achieve high rates of power efficiency, but only POFAME and POFARE are able to complete jobs at rates of 4% and 3%, respectively.

Figures 4.4(c)–4.4(d) show the energy efficiency (Mflops/Joule) E_M and working efficiency E_W, respectively. Both graphs show equal shape, despite they measure the amount of useful work done in different ways. While the energy efficiency measures the useful work to the energy consumed, the working efficiency quantifies the useful work based on the consumed power. In both graphics, for the average task length to MTBF ratio of 0.01, all algorithms achieve E_M and E_W lower than that for the ratio of 0.1 because the same job completion rate is achieved in both cases (Figure 4.4(a)), but the consolidation is lower for the ratio of 0.01 (Figure 4.4(b)).

A comparison of the performance results obtained by CBFIT with those obtained by POFAME and POFARE shows that the overhead imposed by proactive migrations resulted in higher rates of completed jobs, power, energy, and working efficiencies and, therefore, increased the system availability. The number of migrations, as a percentage of the total number of tasks, for the ratios



(a) Completion rate of user' jobs.

(b) Power efficiency.

(c) Energy efficiency.

(d) Working efficiency.

Figure 4.4: Impact of the average task length to MTBF ratio in the performance of scheduling algorithms, without dynamic consolidation, for random workloads. The configuration is: (i) MTBF = 200 minutes; (ii) average failure prediction accuracy = 75%.

of 0.01 and 0.1 are 0.1% and 3.1% for POFAME and 0.1% and 3.8% for POFARE, respectively. For the ratios of 1 and 10, the number of migrations increases to 126% and 350% for POFAME and POFARE, respectively, which is expected because the tasks are longer than the MTBF. However, POFAME and POFARE still complete more jobs with better efficiency than OBFIT, CBFIT, and PBFIT.

4.5.3 Simulation with Google Tracelogs-based Workloads

The goal of using workloads based on Google cloud tracelogs is to evaluate the performance of the algorithms for a realistic workload. The study was performed in two steps: (i) without the consolidation mechanism activated with results shown in Figure 4.5; and (ii) with the consolidation mechanism activated with results presented in Figure 4.6. Likewise Fu [Fu10], the study considers the influence of the failure prediction accuracy (FPA) in the performance of the algorithms. FPA is defined as the difference between the actual and predicted failure time, expressed as a percentage. For example, if a physical node is supposed to fail at minute 100, the failure predictor tool will predict the failure at minute 10 if the FPA is 10%. This means that failure prediction inaccuracies

have a direct impact on the MTBF perceived by the algorithms that consider nodes reliability. As before, the MTBF was set to 200 min. The CBFIT algorithm obtains constant results for all FPA, since it does not take reliability of nodes into account, re-initiating tasks in spare nodes when a node fails.

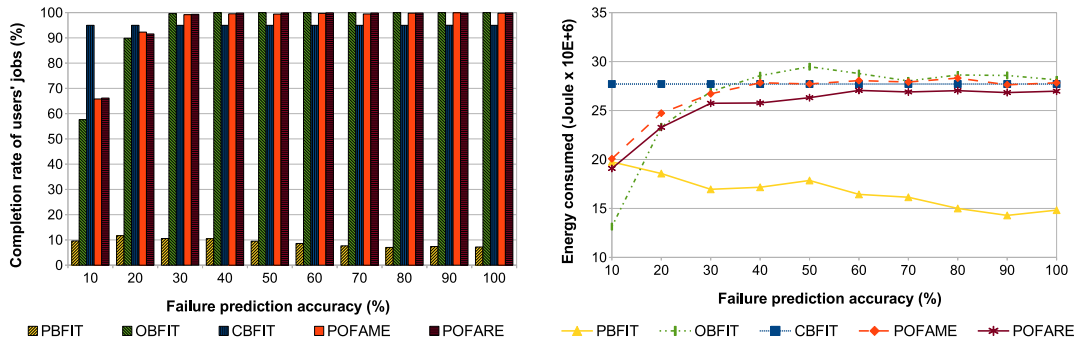
Without Consolidation Mechanism Activated and Varying FPA

The completion rate of users' jobs as the FPA varies from 10% to 100% is depicted in Figure 4.5(a). The configuration is: (i) average task length = 3 minutes; (ii) MTBF = 200 minutes. The first observation is that a simple best-fit algorithm (i.e., CBFIT) is more suitable to complete more jobs for FPA below or equal to 20%. Low values of FPA generates excessive task migrations than required for failure-aware algorithms OBFIT, POFAME, and POFARE, which affects the rate of completed jobs. Therefore, the energy consumed by OBFIT, POFAME, and POFARE (Fig. 4.5(b)) is lower for an FPA below or equal to 20% because when a job cannot be finished, the algorithms do not launch their tasks and do not spend energy computing that part of the job. For an FPA greater than or equal to 30%, OBFIT, POFAME, and POFARE achieve a job completion rate near 100%, which is in accordance with Figure 4.4(a) for the equivalent ratio of 0.01 (in this case, it is 0.015) and an FPA of 75%. Figure 4.5(b) shows that POFARE is generally the best algorithm to produce work at lower energy consumption for different values of FPA. PBFIT provides the worse completion rate of users' jobs because most of the tasks are short, which implies that they would be scheduled only if the physical nodes were failing all the time. Because PBFIT is limited to very few PMs, the energy consumption is also low (Figure 4.5(b)).

The energy efficiency in Figure 4.5(c), and the working efficiency in Figure 4.5(d), provide a better evaluation of the algorithms performance. CBFIT is only best for an FPA of 10%. PBFIT presents the best result for both energy and working efficiencies. For a FPA of 75% (indicated by Fu [Fu10] as the average FPA achieved by the failure predictor tool), POFARE yields improvements of 6.7% and 4.8% over OBFIT, for energy efficiency and working efficiency, respectively. Another important characteristic of POFARE is that it achieves an almost constant energy consumption, energy efficiency, and working efficiency, indicating that the performance and the system availability are nearly independent of the failure prediction accuracy. These results position POFARE as the most suitable algorithm to allocate resources to execute cloud users' jobs in an energy efficient manner.

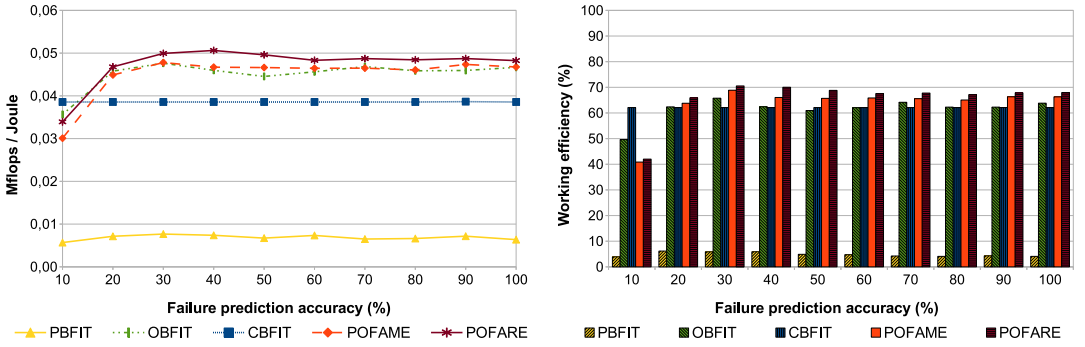
With Consolidation Mechanism Activated and Varying FPA

With the consolidation mechanism activated (Algorithm 1, line 6), VMs are migrated to optimize the system's energy efficiency. Figure 4.6(a) depicts the impact of consolidation mechanism on job completion rate compared with the results obtained without applying the consolidation mechanism (Figure 4.5(a)). The impact on POFARE is below 0.4%, except for the case of FPA of 10%. Comparing the results of Figure 4.6(b) to Figure 4.5(b), POFAME, POFARE, and CBFIT clearly reduce the consumed energy, thus benefiting from power-oriented migrations. Considering an FPA of 75% and POFARE, the energy consumption is reduced by almost 11.3%, without significantly



(a) Completion rate of jobs regarding Failure Prediction Accuracy.

(b) Average energy consumed regarding failure prediction accuracy.



(c) Energy efficiency regarding failure prediction accuracy.

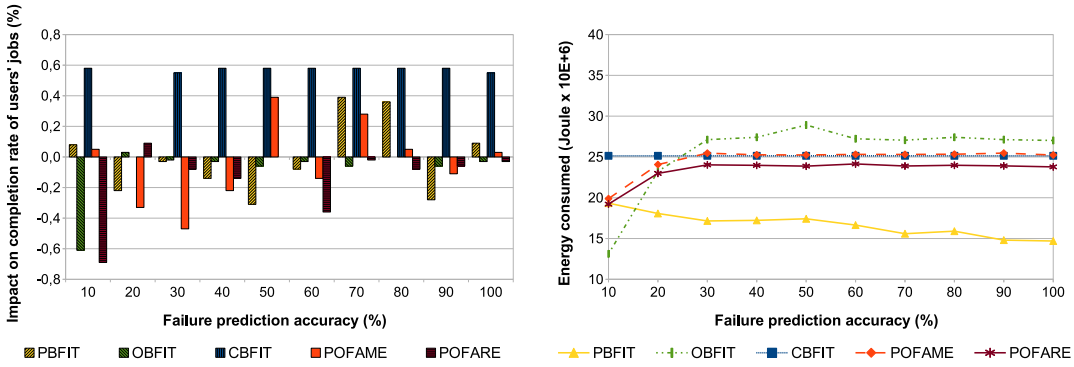
(d) Working efficiency regarding failure prediction accuracy.

Figure 4.5: Energy- and failure-aware scheduling algorithms, without dynamic consolidation, for Google-based workloads. The configuration is: (i) average task length = 3 minutes; (ii) MTBF = 200 minutes.

affecting the job completion rate, which was reduced by less than 0.08%. Figures 4.6(c)–4.6(d) confirm POFAME and POFARE as the algorithms that benefit most from the energy optimizing mechanism. In conclusion, consolidation improves POFARE performance, which outperforms OBFIT in approximately 12.9% and 15.9% with respect to energy efficiency and working efficiency, respectively.

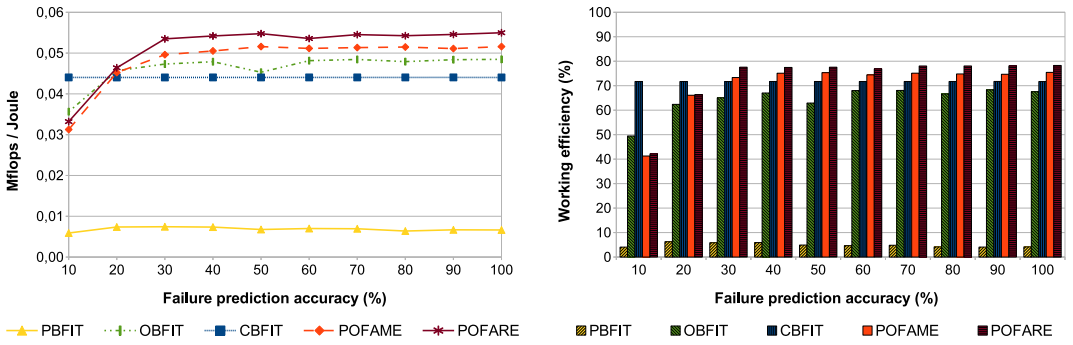
With Consolidation Mechanism Activated and Fixing FPA

Table 4.1 shows the results for the Google cloud tracelogs and an FPA fixed in 75%, with (wcs) and without (wocs) consolidation mechanism. The results show that with the consolidation mechanism, the energy consumption is reduced whilst the rate of finished jobs is improved, for all the algorithms. In particular, POFARE reduces the energy consumption in 11.2%, from 26.9 to 23.9 MJ, and keeping the same job completion rate. Consolidation of VMs implies increasing the number of migrations from 0.8%, due to failure tolerance migrations, to 2.55%, which also includes power-oriented migrations. Moreover, POFARE achieves higher level of consolidation because



(a) Impact in the completion rate of jobs by applying energy optimization.

(b) Average energy consumed regarding failure prediction accuracy.



(c) Energy efficiency regarding failure prediction accuracy.

(d) Working efficiency regarding failure prediction accuracy.

Figure 4.6: Energy- and failure-aware scheduling algorithms, with dynamic consolidation, for Google-based workloads. The configuration is: (i) average task length = 3 minutes; (ii) MTBF = 200 minutes; (iii) $\tau = 55\%$; (iv) $\gamma = 3$.

the algorithm is more relaxed in terms of the power assigned to tasks, even though feasible in modern systems comprising 64 GB of memory.

Table 4.2 registers the results for Google cloud tracelogs with and without consolidation mechanism, but in the absence of node failures. The numbers confirm POFARE as the best algorithm, even in the case of reliable nodes, consuming less power for the same rate of completed jobs.

4.5.4 Experiments in Real Cloud Testbed

The real experiments took place in the testbed introduced in Section 4.4.5. The characteristics of the jobs follow those used in simulations. The experiment reported in this section provides preliminary results of the adoption of the proposed algorithms in a real platform. The configuration of the experiments are: (i) average task length = 3 minutes; (ii) MTBF = 200 minutes; (iii) $\tau = 55\%$; (iv) $\gamma = 3$. Figure 4.7 shows the results obtained from the experiments when applying

Table 4.1: Results for Google cloud tracelogs for energy, job completion rate, ratio of VM migrations to total number of tasks, and number of VMs per PM for the case of 75% of failure prediction accuracy without consolidation (wocs) and with consolidation (wcs).

Algorithm	Energy (MJ) wocs / wcs	E_J (%) wocs / wcs	VM Migrations (%) wocs / wcs	Av. # of VM p/PM wocs / wcs
CBFIT	27.7 / 25.1	95 / 95.5	(3.12) + 1.6	16 / 17
OBFIT	28.3 / 27.2	100 / 100	0 / 0.99	17 / 16
POFAME	28.1 / 25.3	99.6 / 99.8	1.05 / 2.95	17 / 17
POFARE	26.9 / 23.9	99.8 / 99.8	0.8 / 2.55	21 / 22

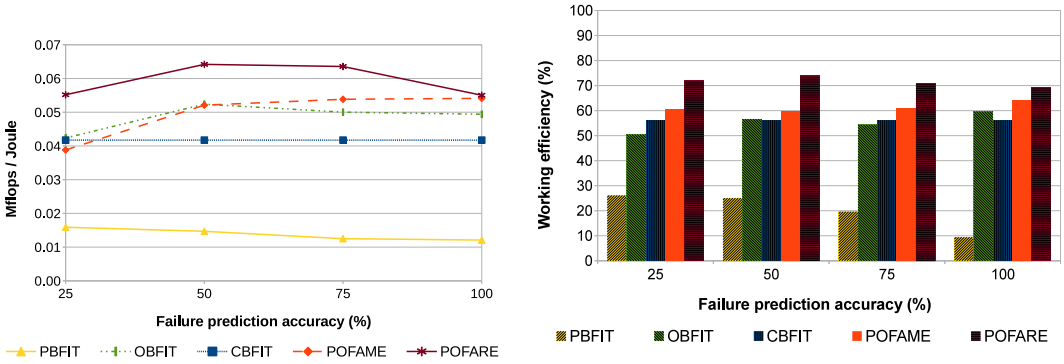
Table 4.2: Results for Google cloud tracelogs without failures for energy, job completion rate, ratio of VM migrations to total number of tasks, and number of VMs per PM, without consolidation (wocs) and with consolidation (wcs).

Algorithm	Energy (MJ) wocs / wcs	E_J (%) wocs / wcs	VM Migrations (%) wocs / wcs	Av. # of VM p/PM wocs / wcs
CBFIT	27.8 / 25.2	100 / 99.8	0 / 2.10	16 / 17
OBFIT	27.7 / 25.2	100 / 99.9	0 / 1.99	16 / 17
POFAME	28.6 / 25.3	100 / 99.8	0 / 2.43	16 / 17
POFARE	26.9 / 23.9	100 / 99.9	0 / 1.82	21 / 22

both energy optimizing and fault-tolerant mechanisms. These results confirm, in general, the POFARE as the best algorithm that optimizes the energy to produce work, as indicated by simulation (Figures 4.6(c)–4.6(d)). Figure 4.7(a) shows that for an FPA of 100%, the delivered Mflops/Joule of the POFARE algorithm decreases to a value near POFAME. This behaviour is justified by the fluctuation of the assigned amount of CPU to each task, which is not constant in the real system, as considered in the simulation. For lower values of FPA, this effect does not affect POFARE because the error margin used to avoid failures accommodates such fluctuations. In Figure 4.7(b), the results for the working efficiency also demonstrate that for different values of FPA, POFARE outperforms POFAME and OBFIT, by 15.2% and 23.5%, respectively. In conclusion, the experiments in real cloud testbed showed that POFARE consumes less energy than any of the other strategies and that completes a similar number of jobs by their deadlines for FPA values equal to or greater than 25%.

4.6 Conclusions

To maximize profit, cloud providers need to apply energy-efficient resource management strategies, and implement failure tolerance techniques. Dealing with failures not only increases the



(a) Energy efficiency regarding failure prediction accuracy.

(b) Working efficiency regarding failure prediction accuracy.

Figure 4.7: Experiments for Energy- and failure-aware scheduling algorithms, with dynamic consolidation, for Google-based workloads. The configuration is: (i) average task length = 3 minutes; (ii) MTBF = 200 minutes; (iii) $\tau = 55\%$; (iv) $\gamma = 3$.

availability of a system and services perceived by cloud users, but also has serious implications in energy waste. However, reducing energy consumption through consolidation and providing availability against failures is not trivial, as it may result in violations of the agreed SLAs.

Two algorithms were proposed, POFAME and POFARE, that apply proactive fault tolerance, to cope with node failures, and consolidation to optimize energy efficiency. The objective is to simultaneously increase the amount of useful Mflops processed per energy unit, as well as the number of jobs completed by the consumed power, in cases where the infrastructure nodes are subject to failure. The algorithms use two different methods to provide energy-efficient and high-available virtual clusters to execute tasks within their deadlines. While the POFAME algorithm tries to reserve the maximum required resources to execute tasks, POFARE sets up the cap parameter from the Xen credit scheduler to execute tasks with the minimum required resources.

In order to analyse the performance of the proposed algorithms, simulations of a cloud computing infrastructure were conducted, by injecting two sets of synthetic jobs. The first set was generated based on the Poisson distribution aiming at assessing the scheduling algorithms performance using different average tasks length to MTBF ratios. The improvement in energy efficiency of POFARE over OBFIT is 23.6%, 16.9%, and 72.4% for the average task length ratios of 0.01, 0.1, and 1, respectively. The improvement in working efficiency of POFARE over OBFIT is 26.2%, 20.3%, and 219.7% for the ratios of 0.01, 0.1, and 1, respectively. The characteristics of the second set of workloads followed the latest version of the Google cloud tracelogs. The results show that POFARE has the best performance, by improving the work per Joule ratio by approximately 12.9% and the working efficiency by 15.9% compared with the OBFIT results obtained with dynamic optimization, and maintaining similar levels of completed jobs. The results also showed that a relaxed strategy (POFARE), which assigns the minimum required resources to each task, yields better results than a strategy (POFAME) that tends to assign the maximum required

resources to each task. These results were then confirmed by tests carried on a real platform.

Chapter 5

Mitigating Performance Deviations in Energy-efficient Virtual Clusters

Colocating multiple executing applications on a single node is a very common technique in cloud computing. User applications are usually consolidated in physical servers, to improve utilization of hardware resources. The aim is to reduce operational costs and maximize profit, by optimizing the energy efficiency. Virtualization is one of the main core technologies in modern data centers, assisting in multiplexing hardware resources for different user applications. However, current virtualization techniques do not provide proper performance isolation, meaning that one VM can generate performance interference to other VMs, deviating applications from specific QoS requirements. This chapter presents an approach to detect and mitigate performance deviations in two different types of applications, with specific QoS constraints. The applicability of the proposed solution is shown by extensive evaluation through simulations for a set of workloads based on the latest version of the Google cloud tracelogs.

5.1 Introduction

THE interest towards cloud computing is growing rapidly. As a result, cloud systems are progressively being adopted in different scenarios, such as business applications, social networking, scientific computation and data analysis experiments [VPB09, KMB+11, SJNL10]. In this regarding, current clouds host a wider range of applications with diverse resource needs and QoS requirements.

Typically, consumers detail the required service level through QoS parameters, which are described in SLAs established with providers [MBS13]. Thus, SLAs are a key element to support and empower QoS in cloud environments. Providing QoS guarantees in current cloud data centers is a very difficult and complex task due to dynamic nature of the environment and richness of application workload characteristics. The problem becomes even more complicated when considering efficient resource usage and technological limitations. Balancing QoS guarantees with efficiency

and utilization becomes extremely challenging, because virtualization does not guarantee performance isolation between VMs, as stated in Section 2. Furthermore, different applications demand for different amount of resources and for diverse QoS requirements. For example, non-interactive batch jobs tend to be relatively stable in resources demanding and QoS defines a completion time, while transactional web applications tend to be highly unpredictable and bursty [GTGB14], and QoS concerns with throughput guarantees.

This study proposes a mechanism to manage resource allocation within a data center that runs CPU- and network I/O-bound applications. The study first delivers a mechanism that estimates the slowdown in co-hosted deadline-driven CPU-bound applications due to contention in on-chip resources, and a second mechanism that responds to changes in demand of network-bound applications. A scheduler algorithm is also proposed to compensate deviations from required performance in both types of applications that apply readjustments in the VMs to PMs mapping to correct such performance deviations. At the same time, the refinement in the assignment of resources to VMs considers the optimization of energy efficiency of the underlying infrastructure. Contrasting with the studies discussed in Chapter 2, the strategy in this thesis proposes to tackle the resource allocation problem within a data center that runs CPU- and network-bound applications with different resource needs and diverse QoS constraints. CPU-bound tasks suffer from performance interference, which causes slowdown in their execution, while network-bound workloads are highly unpredictable and bursty, thus leading to scarcity or wastage of resources. The SLA imposed for each workload is satisfied if: (1) CPU-bound tasks meet their deadlines; and (2) network-bound applications do not experiment scarcity of resources. The strategy improves energy efficiency by avoiding wastage of resources. The main contributions of this study are:

1. A scheduling strategy for heterogeneous workloads in virtualized data centers, with different characteristics and specific SLAs requirements;

2. A mechanism to detect deviations in applications' performance requirements, even when performance data samples contain noise;

3. Support for auto-scaling of resources in order to satisfy SLAs and cope with peak time demands;

4. Support for mitigation of performance degradation caused by contention due to sharing of physical resources among co-hosted applications;

5. Optimization of the energy efficiency by maximizing the utilization of cloud resources;

6. A study about the trade-off between the two opposite objectives: energy efficiency versus performance improvement.

The rest of the Chapter is organized as follows: Section 5.2 establishes the foundation of the proposed cloud computing manager, by defining the mechanisms for detection of power inefficient nodes and performance deviations in heterogeneous workloads having diverse QoS requirements.

Section 5.3 describes PIASA, a power- and interference-aware scheduling algorithm for heterogeneous workloads with diverse QoS requirements. Section 5.4 introduces the metrics used to evaluate the performance of the proposed mechanisms, the characteristics of the workloads in terms of resource needs and performance interference, and describes a state-of-the-art scheduling algorithm for mixing workloads for comparison purposes. The simulation scenario is also characterized in this section. The results and discussion are done in Section 5.5. Section 5.6 is the final conclusion of the chapter.

5.2 An Approach to Power- and Interference-aware Virtual Clusters

The cloud management system considered in this chapter, and its key components involved in performance estimation, power efficiency, and scheduling of applications are illustrated in Figure 5.1. It is based on the architecture introduced in Section 4.2.1, such that a PM is characterized by a CPU capacity C, memory capacity R, LAN network bandwidth N_1 and Internet network bandwidth N_2, access to a shared storage space S for storing the disk images of the VMs, such that $m_i = \{C, R, N_1, N_2, S, F_i\}$. In this study, the aim is to detect and mitigate performance deviations during applications runtime. In this regarding, the MTBF F_i for a PM i is assumed large enough to accommodate applications, and consequently it is not considered. Furthermore, the "Reliability" component in Figure 4.1, is now replaced by a "Performance deviation" component. The eventual integration of the three blocks "Power efficiency", "Reliability", and "Performance deviation", will enable the construction of power-, failure-, and interference-aware virtual clusters. While the local link N_1 is used for resource management purposes, i.e. virtual machine migration, signalling and control data, etc., the link N_2 is utilized by cloud users to access cloud services. Stressing N_1 manifests itself in the overhead of VM migration, while N_2 is directly utilized by users to access deployed services running in VMs.

A VM encapsulates one single application or task, and it is the unit of migration in the system. As described in Figure 5.1, the system may run two possible types of workloads, according to the most consumed resources: CPU-bound or network I/O-bound, each having different behaviour, characteristics, and QoS requirements. A VM runs on top of one PM at a time, and a physical server can consolidate several VMs. The two modules "Power efficiency" and "Performance deviation" estimator continually monitor servers and running applications, while the cloud manager takes decisions to keep the system energy-efficient and fulfilling application's QoS requirements.

5.2.1 Applications Overview

Current cloud data centers host a wider range of applications, with diverse resource needs and different SLA requirements. This section describes the two types of applications considered in this study.

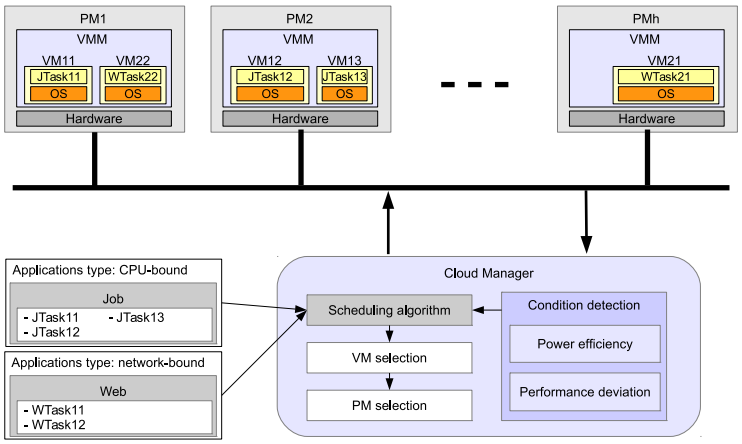


Figure 5.1: Private cloud management architecture.

CPU-bound Workloads

The execution of scientific applications can be modelled as the problem of dynamically scheduling non-interactive batch jobs with specific CPU needs. This problem was already properly modelled in Section 4.3.1, on page 57. For the sake of clarity and readability, the description of CPU-bound workloads is summarised here. Each job j comprises a set of n independent CPU-bound tasks. The QoS requirement for batch jobs defines a deadline d_j, which is defined by their longest task. Each task t has heterogeneous resource requirements, expressed in Mflops. The deadline d_t of a task defines the amount of extra time to complete the task (i.e., the slack time), and constrains the minimum amount of resources $min\ r(t)$ to assign (see Equation (4.3), on page 57). Likewise, a task can consume at most a maximum amount of resources $max\ r(t)$ to run at maximum speed [SVC12]. Tasks with negative slack time are cancelled, and the corresponding jobs are considered incomplete, generating an SLA violation. In order to fulfil the deadline constraint given by Equation (5.1), and hence satisfy the strict SLAs imposed on each job, the CPU power assigned to a task must be between $min\ r(t)$ and $max\ r(t)$.

$$FT(t) \leq d(t) \tag{5.1}$$

where FT is the completion time of the task. The resources initially assigned to co-hosted tasks may need to be adjusted during runtime due to interference caused by sharing of on-chip resources (e.g., CPU, LLC, and memory bandwidth), which induces slowdown, and ultimately impose a deviation between expected and delivered QoS.

Network-bound Workloads

Web workloads are characterized by their dynamic behaviour and bursty resource demanding. Mars et al. [MTS+12] refer that one way to measure the QoS of web servers is to account for the maximum number of successful queries per second, as it is the case of Google's web search. According to Mei et al. [MLP+13], the performance of web servers is CPU-bound under a mix of

small size files, and is network-bound under a mix of large files. This work considers the second case, by assuming that web workloads are mainly characterized by network resource consumption (i.e., network-bound workloads) and have residual CPU consumption. Demand of resources changes abruptly with time. The amount of transferred network I/O data, $b(t)$, served by a task t, at each time instant is given by Equation (5.2).

$$b(t) = \sum_{y=1}^{y=req/tu} \kappa_y \tag{5.2}$$

where κ is the file to transfer in each request y. For each task, the number of requests per time unit req/tu varies with time, and the size of the file to transfer in each request is assumed to be large and practically constant. The SLA defines the performance requirement of web applications in terms of requests served within a period of time. The performance requirement can be translated into CPU capacity $r(t)$, and network I/O bandwidth $b(t)$ [CSW+08]. The objective is to scale applications dynamically and to provide resources to network-bound applications according to fluctuations in workload demands. A request is not served by the web application if the CPU $r(t)$ or network bandwidth $b(t)$ capacity allocated to the application is less than the required capacity at time μ. A SLA violation occurs if a certain number of non-served requests, from the total of requests submitted to the web application within a time interval req/tu, exceeds the maximum specified in the contract.

5.2.2 Power Efficiency Logic

The solution presented in this study proposes to manage virtual clusters that run users applications, in a power- and interference-aware manner. The power optimizing mechanism used to improve power efficiency was introduced in Chapter 4. The objective is to improve the average power efficiency $\overline{E_P}$ of the whole system (see Equation (4.2), on page 53), by migrating VMs away from power inefficient PMs and switch them to low power states.

5.2.3 Performance Deviation Logic

Performance deviations can occur in the two types of workloads considered. In the case of CPU-bound workloads, performance deviation occurs in the form of slowdown due to contention in shared on-chip resources, thus precluding a task to execute within its deadline. Several studies (e.g., [DFB+12, GLKS11, LHK+12]) demonstrated to be possible to determine with an error around 10% the slowdown in the execution of tasks. The slowdown experienced by a CPU-bound task t in the PM i is expressed by Equation (5.3).

$$D_i(t,\mu) = \frac{\frac{W(t,\mu)}{\widetilde{r_i}(t)} - \frac{W(t,\mu)}{r_i(t)}}{\frac{W(t,\mu)}{r_i(t)}} + \varepsilon \tag{5.3}$$

where $r_i(t)$ and $\widetilde{r}_i(t)$ are the expected and effectively obtained CPU resources (in Mflop/s) assigned to the task t, respectively. The parameter $r_i(t)$ can be obtained from state-of-the-art slowdown meters (e.g [DFB+12]), while $\widetilde{r}_i(t)$ is obtained by monitoring local resources, and corresponds to how much CPU is being used. The performance deviation $D_i(t,\mu)$, or slowdown in the case of CPU-bound applications, can be negative (i.e., the task is executing faster) or positive, and the same applies to the estimation error ε (i.e., the average error in slowdown data samples plus random fluctuations).

Performance deviation in network-bound applications can occur due to differential between initially assigned and current demand of resources. The performance deviation is stated by Equation (5.4).

$$D_i(t,\mu) = \frac{b_i(t,\mu) - \widetilde{b}_i(t,\mu)}{b_i(t,\mu)} \tag{5.4}$$

where $b_i(t,\mu)$ and $\widetilde{b}_i(t,\mu)$ are the demanded and effectively assigned network I/O bandwidth (in Gbit/s) at instant μ to task t running in node i, respectively. Due to fluctuations, the performance deviation $D_i(t,\mu)$ can be negative (i.e., the task is demanding for less I/O bandwidth) or positive when the demand exceeds the assigned I/O bandwidth.

5.2.4 Performance Enforcing Logic

The mechanism to detect performance deviations and enforce QoS requirements depends on the type of workload under analysis, since the root of interference differs accordingly. In this sense, the performance enforcing logic is analysed separately for each type of workload.

CPU-bound Workloads

The high variability of workloads in demanding of resources and the unexpected performance deviations due to interference among co-hosted VMs sharing on-chip resources, make application QoS requirements difficult to satisfy. To cope with performance deviations in CPU-bound applications, a module combining two blocks is deployed: (i) Kalman Filter (KF) noise removal; and (ii) Linear Regression-based (LR) completion estimator. The simple Kalman filter [Har90] is a powerful estimator, that has the ability to smooth noisy data and to provide reliable estimates of signals affected by indirect, inaccurate and uncertain observations (e.g., Gaussian noise). In turn, regression analysis [SL12] is a common used statistical procedure to model relationships between variables, with the purpose of predicting dependent variable Y on the basis of explanatory variable X, such that $Y = aX + b$. The parameters a and b represent the slope and the intercept, respectively. The module is shown in Figure 5.2.

The module works by injecting slowdown data samples for a task t, provided by upstream state-of-the-art tools (e.g., [DFB+12]), into the KF block to eliminate (Gaussian distributed) unpredictable disturbances in slowdown samples. Then, the filtered slowdown data samples feed the LR completion estimator to predict the tasks completion time. The LR starts by determining the remaining

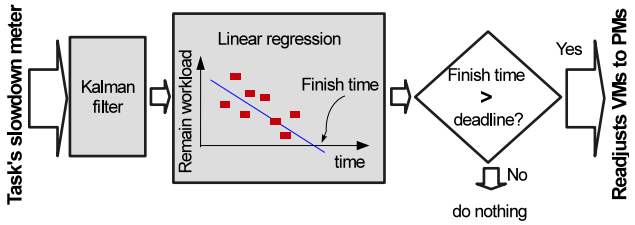


Figure 5.2: Performance deviation estimator module for CPU-bound workloads.

workload for the last ψ sampling intervals based on filtered ψ slowdown data samples and monitored task CPU utilization. These ψ remaining workload estimates are then used to fill Y axis, and the ψ time instants in which these estimates are observed are registered in the X axis. After, based on the LR equation (i.e., $Y(t) = aX(t) + b$), $X(t)$ is determined for $Y(t) = 0$ (i.e., the time instant when there is no remaining workload to process): $X(t) = -b/a$. So, $X(t)$ represents the time instant in which task t is predicted to accomplish. If $X(t) > d_t$ then the task t is predicted to finish after the deadline d_t, incurring a SLA violation. Henceforth, the combination of KF and LR will be referred as KFLR module.

Network-bound Workloads

The handling of performance deviations in network-intensive workloads is achieved based on the definition of a soft limit and hard limit of resources to assign to each VM. The hard limit implements an upper bound of resources consumption, which is determined by the total amount of free resources in the PM. The soft limit implements a lower bound of bandwidth and CPU resources availability. Thus, in case of resources contention among collocated VMs or overloaded PM, the soft limit establishes a minimum of resources guaranteed for each VM. For example, if 500 Mbit/s of network bandwidth are assigned to a VM_1 (i.e., the soft limit is 500 Mbit/s), and there is a co-hosted VM_2 that requires only 300 Mbit/s at a certain time interval, then VM_1 can use the remaining 200 Mbit/s from the total 1 Gbit/s deployed by the PM. The same applies for the CPU resource. The soft limit in CPU and network bandwidth is stipulated by the output of two separate Kalman filters (KFIO), one per type of resource, which receive as input CPU and network bandwidth usage samples. Figure 5.3 shows the scheme of the Kalman filters. If the output of one of the Kalman filters deviates $\delta = 5\%$ from the assigned amount of resources (i.e., from the soft limit), then the scheduling algorithm is invoked to proceed with an adaptation. More or less resources can be assigned to the VM as the estimated soft limit is under or above the required resources. The aim of the strategy is to avoid over-provisioning and consequent wastage of resources, as well to make sure that a VM can access more resources in case of abrupt demand. The scheduling algorithm considers only the estimated soft limits.

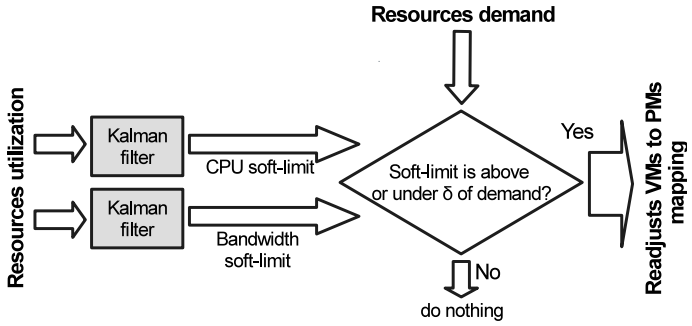


Figure 5.3: Performance deviation estimator module for network-bound workloads. The parameter δ is the threshold below/above which the scheduling algorithm is invoked.

5.3 PIASA - A Power- and Interference-Aware Scheduling Algorithm

The PIASA scheduling algorithm is based on POFARE algorithm, properly introduced in Chapter 4. PIASA algorithm considers eventual performance deviation in different types of workloads, with diverse QoS requirements. Because the problem of mapping the VMs to the PMs is NP-complete, the proposed algorithm is heuristic. The problem of scheduling and executing workload applications is divided in two smaller problems: (1) VM selection; and (2) VM placement. The PIASA algorithm is described in Algorithm 3.

5.3.1 VM Selection

In order to select a VM to schedule, the algorithm starts by creating a list of tasks $\{t_1,\ldots,t_{l\times k}\}$, grouped according to the reason of scheduling. Each group is associated with a scheduling priority, from the highest to the lowest: (1) performance deviation; (2) new tasks; and (3) low power-efficient PM. In the first group (i.e., performance deviation), network-intensive tasks have priority over CPU-intensive tasks, being that priority inverts for other groups. The aim is to exploit the deadline of CPU-bound workloads to implement adjustments in the VMs to PMs mapping regarding power efficiency. The same applies for scheduling new tasks, since deadlines start counting as soon jobs are submitted. This strategy provides network-intensive applications more stability and responsiveness by avoiding migration overheads, and consequently serving more requests. Network-intensive tasks within groups are sorted in ascending order according to the difference between demanded and assigned resources (i.e., $D_i(t,\mu)$ in Equation (5.4)). The next cases hold for $D_i(t,\mu)$: (1) it will be negative when demand of resources is inferior than assigned resources; (2) it will be zero when is a new task; (3) it will be positive if demand exceeds the assigned resources. By shrinking first VMs with excess of capacity ($D_i(t,\mu) < 0$), applications suffering from scarcity of resources ($D_i(t,\mu) > 0$) can expand and see their QoS re-established without migration. In turn, CPU-intensive tasks in each group are sorted in ascending order according to their slack

time (i.e., according to Equation (4.4), on page 57). Tasks with negative slack time are eliminated from the list. This logic is implemented in lines 3–6 of the Algorithm 3.

5.3.2 VM Deployment

The algorithm follows by picking up the first task t from the prioritized list in line 7 of Algorithm 3. If task t is in the list due to performance deviation (line 8), the algorithm proceeds by expanding or shrinking the resources assigned to the VM running it (line 9). If expansion of VM shows to be impossible due to scarcity of free resources in the PM, then the algorithm proceeds in different ways according to the type of workloads. If task t is CPU-bound, the algorithm tries to migrate the VM running it in line 11. If the migration overhead is greater than the slack time in line 14 (i.e., the task will not finish within the deadline), the algorithm tries to migrate co-located CPU-bound tasks (lines 15–19), starting by the one with the biggest slack time. This process is repeated until released resources make expansion become possible in line 20. If the task is network-bound, the strategy is to migrate CPU-bound workloads (lines 15–19) in an attempt to free resources. If even though, freed resources are not enough so network-bound application can expand in line 20, the algorithm tries to migrate the application in line 25. The function *getNewPM* retrieves the PM i that provides the highest value for the interference-power metric ip, as given by Equation (5.5).

$$ip = max\{Value(i), i = 1..h\} \tag{5.5}$$

where h is the number of PMs in the infrastructure. In turn, the $Value(i)$ is given by Equation (5.6).

$$Value(i) = \begin{cases} E_{P_i} & \text{if } \sum_{t=1}^{n-1} r_i(t,\mu) < \alpha \\ I_i & \text{otherwise} \end{cases}, \tag{5.6}$$

$$\text{where } I_i = \begin{cases} 1 - \frac{\bar{D}_i(\mu)}{max\,\bar{D}_h(\mu)} & \text{if workload is CPU-bound, and } \bar{D}_i(\mu) > 0 \\ \Phi_i(\mu) & \text{otherwise} \end{cases}$$

where I_i is the interference factor on machine i, E_{P_i} is the power efficiency defined by Equation (4.1), on page 53, $\bar{D}_i(\mu)$ is the average slowdown experienced by $\{t_1,\ldots,t_{n-1}\}$ tasks co-hosted in PM i (see Equation (5.8)), $max\,\bar{D}_h$ corresponds to the maximum slowdown occurring in some of h total PMs in the data center, and α is the weight put on each factor in making decisions on the PM to choose. As α approaches 1, ip becomes more power-aware biased, and the algorithm tends to select the most power-efficient node to run the task at expenses of higher number of SLA violations. Instead, as α tends to 0, ip gets interference-aware biased, and the algorithm attempts to find the node with the least interference among the qualified candidates, thus providing better conditions to fulfil the task's QoS requirements. The value of α can be dynamically adjusted according to the way application QoS deviations evolve and the cloud provider

Algorithm 3 Power- and Interference-Aware Scheduling Algorithm (PIASA).

```
1:  function PIASA(pmList, taskList)
2:      map ← NULL
3:      taskList.removeTasksHavingNegativeSlackTime()
4:      taskList.groupTasksByReason()
5:      taskList.sortTasksInGroupsByDemandDifference(net_type)
6:      taskList.sortTasksInGroupsByIncreasingSlackTime(cpu_type)
7:      for all task ∈ taskList do
8:          if task.reasonIsPerformanceDeviation() then
9:              if getNewProvision(task, map) == FALSE then
10:                 if task.getTaskType() == cpu_type then
11:                     getNewPM(pmList, task, map)                    ▷ try migration
12:                 end if
13:             end if
14:             while map.findTask(task) == FALSE do                   ▷ no candidate PM
15:                 tempTask ← task.getNextCohostedTask(cpu_type)
16:                 if tempTask == NULL then
17:                     break
18:                 end if
19:                 getNewPM(pmList, tempTask, map)          ▷ tries to migrate co-hosted VM
20:                 if getNewProvision(task, map) == TRUE then
21:                     break
22:                 end if
23:             end while
24:             if task.getTaskType() == net_type and map.findTask(task) == FALSE then
25:                 getNewPM(pmList, task, map)
26:             end if
27:         else
28:             getNewPM(pmList, task, map)
29:         end if
30:         if map.findTask(task) == TRUE then
31:             applyMap(map)
32:         end if
33:         taskList.removeTasks(map)
34:     end for
35: end function
```

objectives. The parameter $\Phi_i(\mu)$ expresses the factor of consolidation and interference, and is defined by Equation (5.7).

$$\Phi_i(\mu) = 1 - \frac{(n-1) - |H_i(\mu)| + \#Loads_{t_n}}{2 \times V}, \quad \text{and} \quad H_i(\mu) = \#Loads_{IO} - \#Loads_{CPU} \tag{5.7}$$

where μ is the current time instant, $(n-1)$ is the number of VMs running in server i, $H_i(\mu)$ is the heterogeneity degree, and $\#Loads_{t_n}$ is the number of loads of the same type as the task t_n running in

the PM i. The $\#Loads_{CPU}$ and $\#Loads_{IO}$ are the number of CPU-bound and network-bound loads running on machine i, respectively. V equals the number of VMs hosted in the PM running more VMs. For example, if at time instant μ a server hosts 3 CPU-bound tasks, and 1 network-bound task, and a new CPU-bound task is submitted for scheduling, then $H_i(\mu) = -2$ and $\#Loads_{t_n} = 3$.

In this regarding, the member $\Phi_i(\mu)$ will vary within [0, 1]. As $\Phi_i(\mu)$ tends to zero, for a given machine i, its $Value(i)$ reduces, as well as the chance to be selected to run task t_n. $\Phi_i(\mu)$ tends to zero when: (1) the number of VMs $(n-1)$ hosted in the server i increases, so that the system will give preference to less loaded machines; (2) the heterogeneity degree decreases, meaning that the machine is balanced since workloads running in the server tend to be of different type; and (3) the type of task t_n is of the same type of tasks running on PM i, i.e. the system will avoid hosting tasks of predominantly the same type. This last assumption follows the fact that the impact in the performance of co-hosted workloads is alleviated if they show dissimilar resource needs [KKB+07, HL13]. The denominator $2 \times V$ forces the normalization of the equation, thus ranging within [0, 1]. Smaller values of $\Phi_i(\mu)$ will result in smaller values of I_i, and consequently of ip, thus becoming unlikely to schedule task t_n in node i. The opposite applies as $\Phi_i(\mu)$ tends to 1.

After the machine with highest $Value(i)$ is selected, designated as PM i, the slowdown that task t_n will be subjected to depends on the load of the machine. For CPU-bound workloads the slowdown $\beta_i(\mu)$ is provided by Equation (5.8).

$$\beta_i(\mu) = \begin{cases} \bar{D}_i(\mu) & \text{if task } t_n \text{ is new} \\ D_i(t_n,\mu) & \text{if shrinking/expanding task} \\ max(D_i(t_n,\mu), \bar{D}_i(\mu)) & \text{if migrating task} \end{cases}, \quad \text{where} \quad \bar{D}_i(\mu) = \frac{\sum_{t=1}^{n-1} D_i(t,\mu)}{n-1} \tag{5.8}$$

where the value of slowdown $\beta_i(\mu)$ depends on the reason why the algorithm is invoked. If the task t_n is going to be scheduled for the very first time, then $\beta_i(\mu) = \bar{D}_i(\mu)$, where variable $\bar{D}_i(\mu)$ is the average slowdown experienced by $\{t_1, \ldots, t_{n-1}\}$ tasks co-hosted in candidate PM i. If the algorithm needs to shrink/expand assigned resources to task t_n, the task slowdown $D_i(t_n,\mu)$ is considered (see Equation (5.3)). In case of migration, the policy applied is to choose the biggest of the two: (i) slowdown experienced by the task, $D_i(t_n,\mu)$, or (ii) average slowdown of candidate PM i, $\bar{D}_i(\mu)$. Knowing $\beta_i(\mu)$, the scheduling algorithm is now able to estimate the additional amount of CPU resources $\Delta r_i(t_n, \beta_i(\mu))$ to assign to task t_n to mitigate the performance deviation, and which value is dictated by Equation (5.9). This methodology was previously shown by Nathuji et al. [NKG10] as necessary to achieve the performance that cloud customers would have realized if they were running in isolation.

$$\Delta r_i(t_n, \beta_i(\mu)) = \frac{W(t_n,\mu) \times \beta_i(\mu)}{d_{t_n} - \mu - m_{ei}} \tag{5.9}$$

where $m_{ei} = 0$ for new or shrinking/expanding tasks.

In the case of network-intensive workloads, the additional amount of bandwidth resources $\Delta r_i(t_n, \beta(\mu))$ to assign to task t_n is given by Equation (5.10), and it is determined by the difference between the previously assigned resources $\widetilde{b_i}(t_n, \mu)$ and the demand of resources estimated by the Kalman filter $\check{b_i}(t_n, \mu)$, at instant of time μ. The same logic is applied to CPU resource.

$$\Delta r_i(t_n, \beta(\mu)) = \check{b_i}(t_n, \mu) - \widetilde{b_i}(t_n, \mu) \tag{5.10}$$

Next, the algorithm determines the whole amount of resources to fulfil the QoS requirement of tasks during runtime. In the case of CPU-bound workloads, Equation (5.11) specifies that a task t_n can be instantiated in, or migrated to, a PM i at time $\mu + m_{ei}$, if: (i) the node provides the minimum resources $r_i(t_n) + \Delta r_i(t_n, \beta(\mu))$ required by the task, during the $d_{t_n} - \mu - m_{ei}$ time interval, to execute the remaining workload $W(t_n, \mu)$ at instant μ; (ii) and required resources $r_i(t_n)$ are not more than $max\ r(t_n)$ to execute the task by its deadline.

$$\begin{cases} (r_i(t_n) + \Delta r_i(t_n, \beta(\mu))) \times (d_{t_n} - \mu - m_{ei}) \geq W(t_n, \mu), \\ r_i(t_n) + \Delta r_i(t_n, \beta(\mu)) \leq max\ r(t_n) \end{cases} \tag{5.11}$$

Finally, the scheduling algorithm checks if candidate PM i can provide the minimum resources required by task t_n, thus fulfilling Equation (5.12) as follows:

$$r_i(t_n) + \Delta r_i(t_n, \beta(\mu)) + \sum_{t=1}^{n-1} r_i(t) \leq C_i \tag{5.12}$$

where C_i represents the available capacity (i.e., CPU or network bandwidth) in node i. In the end, the algorithm applies a soft limit upon CPU and bandwidth resources assigned to task t_n, which is enough to complete by deadline, in the case of CPU-bound tasks, or to maintain the throughput, in the case of network-bound tasks. As explained before, the soft limit is updated dynamically in accordance to applications QoS requirements.

5.4 Performance Evaluation

The purpose of this section is to establish and describe the scenario of simulations, workloads injected, algorithms tested for scheduling of workloads, and the metrics considered to assess the performance of proposed mechanism to deal with performance deviation and power inefficiencies.

5.4.1 Simulation Setup

The simulated scenario comprised a cloud data center infrastructure aggregating 50 homogeneous physical servers. Each server deploys a CPU with a capacity assumed as 800 Mflops/s. The electric power consumption at full load was 275 W, that is in accordance with the SPECpower benchmark [SPE14] for the first quarter of 2014 for modern servers. The parameters $p1$ and $p2$ in

Equation (4.1), on page 53, were set up to 70% and 30% of full power consumption, respectively. A VM requires a RAM size of 256, 512, or 1024 MB, randomly selected. The bandwidth of both network interfaces for each server was considered to be 1 Gbit/s. Memory ram usage and available network bandwidth in LAN link, N_1, determine the migration overhead of a VM [BAB12, HG09].

During simulation, workloads are submitted to the cloud infrastructure. The scheduling algorithms have no knowledge of when applications arrive, nor about the changing in workload demands. All the tested algorithms reserved 15% of free bandwidth in Internet link, N_2, in every PM to better accommodate abrupt oscillations without migration.

5.4.2 Performance Metrics

The effective performance evaluation and comparison of algorithms implies the definition of performance metrics that capture the relevant characteristics of the algorithms, as well as the performance boundaries as external conditions evolve (e.g., slowdown data samples provided by state-of-the-art upstream slowdown meters, workload variability level, etc.). This section introduces the main metrics to assess the performance of the algorithms in terms of energy efficiency and capacity to fulfil the application's QoS requirements. These metrics are: (i) completion rate of users' jobs; (ii) service rate of transaction requests; (iii) energy efficiency; and (iv) the working efficiency.

The Completion Rate of Users' Jobs

The completion rate of users' jobs metric, E_J, previously defined by Equation (4.9), on page 62, measures the performance of the algorithms in terms of capacity to complete jobs. It corresponds to the SLA metric for CPU-bound workloads.

The Service Rate of Transaction Requests

The service rate of transaction requests metric, E_R, applies to network-intensive workloads, measuring the percentage of SLA fulfilment by relating the number of applications that successfully served, at least, 95% of requests, within a time unit.

$$E_R = \frac{R_C}{R_S} \tag{5.13}$$

where R_C is the number of requests that were successfully served within a time unit, and R_S is the number of submitted requests. Or in other words, R_C is a measure of the average throughput of application at different workload rates. The E_R value falls in the interval [0, 1].

The Energy Efficiency

The energy efficiency metric, E_M, already introduced by Equation (4.10), on page 62, relates the sum of Mflops of the workloads from all tasks of successfully completed jobs by the overall energy consumption, in Joules.

The Working Efficiency

The working efficiency metric, E_W, described by Equation (4.11), on page 62, quantifies the useful work done by the consumed power.

The mechanism introduced in this study proposes to deal with the trade-off between power efficiency and performance interference. Power efficiency can be achieved through higher level of consolidation that, in turn, causes in general more performance degradation and consequently affects negatively the rate of completed jobs.

5.4.3 Workload Characteristics

Because different types of workloads present diverse QoS requirements and demand for different resources, this section describes the characteristics of the two types of workloads considered in this study. The specificities of the performance interference occurred among co-hosted applications is also outlined.

Workloads Based on Google Cloud Tracelogs

Two sets of workloads were created based on key findings from recent comprehensive analysis (e.g. [MGTX14, LC12]) of the workload characteristics derived from Google cloud tracelogs. The first set, properly introduced and described in Section 4.4.3, on page 63, comprised CPU-bound workloads to simulate users' non-interactive batch jobs. The second set included 148 network-bound groups of web applications, comprising a total of 413 tasks. These tasks are mostly network bandwidth consuming. Ersoz et al. [EYD07] refers that lognormal distributions model web server workloads well. In this sense, the required bandwidth due to oscillation in the number of requests per time unit and the file size to download varies randomly with time, following a lognormal distribution with average value and standard deviation of 52.5 and 265, respectively. The CPU consumption changes as well, but in a residual way, following a lognormal distribution with average value and standard deviation of 4 and 3.5, respectively. In both types of workloads, the runtime is specified by the user, and the VMs running them require a RAM size of 256, 512, or 1024 MB, selected randomly.

Performance Interference Characteristics

Co-hosted tasks sharing the server's resources present a behaviour often described as non-deterministic [LHK+12]. However, several studies (described in Chapter 2) contributed with considerable key findings about the characteristics of performance interference among co-located VMs. For example: (1) the interference is larger when two applications of the same type are co-located [PLM+10]; (2) interference among CPU-intensive applications can impose a slowdown up to 120% [GLKS11], or even more [HA12]; (3) the linear relationship between total completion time and individual completion times is lost when different types of jobs co-execute in the system [LHK+12]; (4) very large applications in terms of working set size are not affected by other co-hosted workloads, and small applications do not suffer performance degradation as far the sum

of working set sizes remains inferior to the server's cache [VAN08b]; (5) the degradation in performance does not increase when the pressure over the servers' cache reaches about the double of the capacity [MTS+12].

Based on these key findings, it was simulated performance interference among co-located CPU-intensive workloads. A CPU-intensive application utilizes a certain amount of cache to store data during runtime. Such an amount is determined by a Poisson distribution, with mean of 13% (an acceptable value considering current processors [JMJ06]) of the total physical cache. Poisson distribution has been utilized to study LLC management strategies [LHP15].

The degree of performance interference of an application depends on the application working set size and the working set size of co-hosted applications. Specifically, the characteristics of the performance interference among co-hosted workloads follow the key findings [GLKS11, HA12, MTS+12, VAN08b]: (i) interference among co-hosted applications is residual (e.g., almost zero) as long as the sum of their working set sizes is less than the cache size; (ii) the performance of an application degrades as soon it gets closer to the size of the cache and tends to stabilize afterward, as it moves away from that size; (iii) cache contention decreases CPU throughput (instructions dispatched per second), hence imposing slowdown in the execution of applications. The performance degradation of the applications sharing the same PM is determined by $e^{-((x-1)/2x)^3}$ in the case in which the sum of the working sets size, x, is inferior to the PM cache size; and it is $e^{-((x-1)/2x^{0.95})}$, otherwise. These expressions were inferred from the results presented in [MTS+12, NKG10, VAN08b]. The value x is given in percentage form, so that x will be greater than 100% when application requirements exceed PM capacity.

Since it is assumed that web applications are characterized mainly by network-intensive workloads, the interference among co-hosted web applications is caused essentially by contention in network I/O channel.

5.4.4 Alternative Strategies for Comparison Purposes

The proposed mechanism, comprising performance deviation detectors and a scheduling algorithm, was compared with other state-of-the-art mechanisms to better assess its performance. These alternative mechanisms are listed next.

Performance Deviation Estimators

In order to compare the performance of proposed slowdown estimator, three other common methods were implemented based on data samples analysis performed over a sliding window. The sliding window contained 5 performance deviation samples per minute, taken at constant intervals. The threshold was set up to 10% more of tasks minimum execution time. The first method, Simple ThresholD (STD), triggers an adaptation if 3 slowdown data samples exceed the predefined threshold value. The second method, Average ThresholD (ATD), triggers an adaptation if the average slowdown from samples within the window exceeds the predefined threshold value.

The third approach, Extended ThresholD (ETD), first calculates the average slowdown from samples within 3 consecutive windows and triggers an adaptation if the result exceeds the mentioned threshold.

State-of-the-art Scheduling Algorithms

Garg et al. [GTGB14] proposed an admission control and scheduling mechanism to maximize the resource utilization and profit, at the same time it ensures that the QoS requirements of users are met as specified in SLAs. The algorithm was previously discussed in Section 2.4.2. The performance of this algorithm is compared to PIASA, the algorithm proposed in this study.

5.5 Simulation Results and Analysis

The simulation of the suggested solution is divided in 5 individual steps, in order to extensively assess its performance in terms of capacity to enforce power efficiency and to accomplish with application QoS requirements. These steps are described below.

5.5.1 CPU-bound Workloads

This set of simulations intends to analyse the performance of the slowdown estimator and scheduling algorithm in the provisioning of resources to CPU-intensive jobs.

Performance of the Scheduling Algorithms

The results for the performance of scheduling algorithms, in the absence of interference among co-hosted CPU-bound workloads (i.e., applications do not suffer slowdown during runtime), are registered in Table 5.1. The consolidation mechanism for power efficiency optimization was activated. The α parameter in Equation (5.5) was set to 1.0 in order to make PIASA power efficiency biased and provide a fairer comparison to Garg et al. [GTGB14] which works based on the best-fit manner. The results show that PIASA achieves the same completion rate of jobs as Garg et al. [GTGB14]. However, PIASA improves the energy efficiency and working efficiency in about 20.6%, and 21.8%, respectively, which means that it produces the same work with CPU-bound workloads as Garg et al. [GTGB14], but consuming less energy, as shown in column "Energy (MJ)" of the table. Indeed, PIASA consumes less energy because it achieves higher levels of consolidation due to the use of cap parameter (i.e., the imposed soft limit), and thus using less average number of PMs during simulation, "Avg # of PMs".

In the next set of simulations, performance interference was introduced among co-hosted CPU-bound tasks, and the results are shown in Table 5.2. The consolidation mechanism for power efficiency optimization was activated. The upstream slowdown meter provides noisy slowdown data samples with an average error of 10%, and standard deviation of 20%. The α parameter was set up to 1.0, due to comparison purposes to Garg et al. [GTGB14], and mostly because,

Table 5.1: Performance of the scheduling algorithms in the management of CPU-bound workloads, without interference among co-hosted tasks.

Algorithm	E_J (%)	E_M (Mflops/J)	E_W (%)	Energy (MJ)	Avg # of PMs
Garg et al. [GTGB14]	100.00	0.0572	59.23	114.7	7.61
PIASA	100.00	0.0690	72.15	95.2	6.10

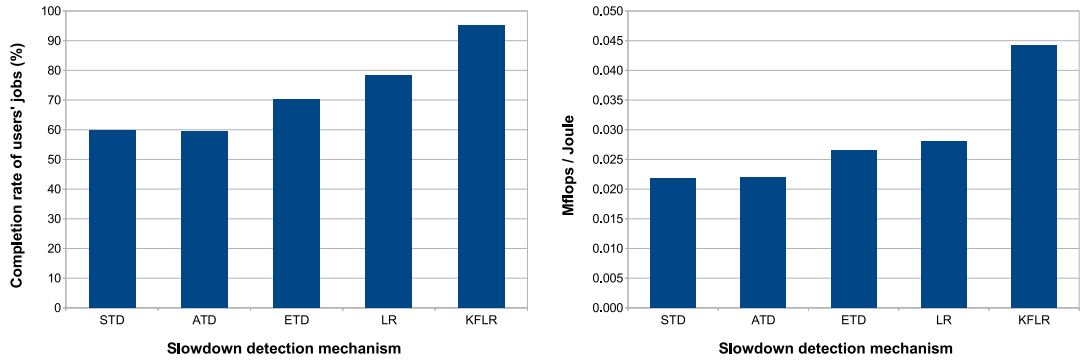
Table 5.2: Performance of the scheduling algorithms in the management of CPU-bound workloads, with interference among co-hosted tasks and applying power optimization (noisy slowdown samples with average error of 10%, and standard deviation of 20%). The α parameter in Equation (5.5) was set to 1.0.

Algorithm	E_J (%)	E_M (Mflops/J)	E_W (%)	Energy (MJ)	Avg # of PMs
Garg et al. [GTGB14]	12.2	0.0023	8.29	114.7	7.61
PIASA	95.60	0.0452	69.21	127.7	7.53

according to Equation (5.5), it relates to the case of optimizing power efficiency. In this regarding, $\alpha = 1.0$ refers to the case in which more performance interference occurs due to a high level of consolidation. The proposed mechanism has to be capable of restoring the QoS required by applications. The PIASA algorithm is combined with the KFLR slowdown estimator to deal with possible slowdown situations in the execution of tasks. The results show that the performance of Garg et al. [GTGB14] algorithm degrades substantially, since it ignores the slowdown in the execution of tasks. In turn, PIASA strategy is able to complete more than 95% of the submitted jobs, E_J, largely outperforming Garg et al. [GTGB14]. Comparing these results with those of Table 5.1, PIASA uses now more 24% of servers during simulation, as indicated by "Avg # of PMs". In fact, due to slowdown in the execution of tasks, more resources are allocated to VMs in order to re-establish the required levels of QoS, implying more servers in active state and thus more energy consumed. The column "Energy (MJ)" confirms an increase of 34%. Furthermore, the average time to complete the tasks increased in 33%, since tasks execute longer due to the slowdown generated. Since more energy is consumed in an effort to produce the same level of work, the energy efficiency E_M, as well as the working efficiency E_W, is reduced compared to the case of absence of slowdown during tasks runtime, as registered in Table 5.2.

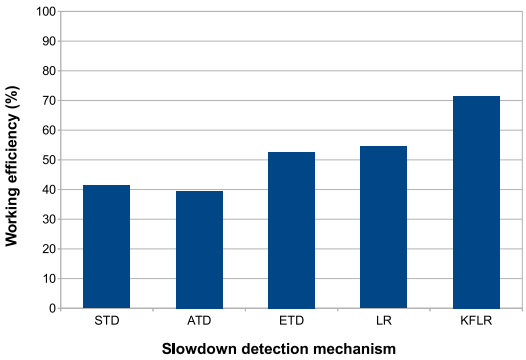
Performance of the Slowdown Estimator

This section analyses the efficacy of the proposed slowdown estimator mechanism introduced in Section 5.2.4. Simulations are carried out by combining different slowdown estimator alternatives introduced in Section 5.4.4 with PIASA algorithm, which performs better as seen above. As before, $\alpha = 1.0$ in Equation (5.5), and the slowdown data samples provided by the upstream slowdown meter have an average slowdown error of 10%, and standard deviation of 20%.



(a) Completion rate of users jobs.

(b) Energy Efficiency.



(c) Working Efficiency.

Figure 5.4: Comparison among slowdown estimator mechanisms in the management of CPU-bound workloads, combined with PIASA scheduling algorithm (noisy slowdown samples with average error of 10%, and standard deviation of 20%). The α parameter in Equation (5.5) was set to 1.0.

Figure 5.4(a) shows that KFLR achieves superior performance than all the other slowdown estimator techniques, by completing almost 96% of jobs, which represents about 17.9% more relatively to LR, and about 37.1% more than so common applied STD method. Both STD and ATD methods are very sensitive to fluctuations in slowdown measurements, responding with almost 38% more of VM migrations comparatively to PIASA, leading to excessive migration overhead. In turn, ETD seems to react too late, triggering VM migrations when tasks are no longer capable of complete by their deadlines. Figures 5.4(b) and 5.4(c) quantify the useful work done by the consumed energy and power, respectively. Figures confirm KFLR as the estimator that performs better, delivering more work, for less energy consumption. These results suggest that KFLR provides higher confidence levels, taking better decisions despite the noisy slowdown data samples.

Impact of ASE in the Performance of PIASA + KFLR

Several state-of-the-art tools are referred to as providing slowdown measurements with an Average Slowdown Error (ASE) around 10% [ZT12, DFB+12, KKB+07], which is the value used so far in

Table 5.3: Performance of PIASA + KFLR in the management of CPU-bound workloads, while varying ASE in data samples provided by state-of-the-art upstream slowdown meters. The α parameter in Equation (5.5) was set to 1.0.

ASE	E_J (%)	E_M (Mflops/J)	E_W (%)	$\bar{\beta}$ (%)
5%	97.29	0.0475	75.41	20.37
10%	95.60	0.0452	69.21	22.30
15%	92.20	0.0413	67.51	25.36
20%	85.50	0.0346	61.68	29.06
25%	73.36	0.0247	51.47	32.32

previous simulations. This section evaluates the efficacy of the combination PIASA + KFLR, for different values of ASE. Table 5.3 shows the evolution of E_J, E_M, E_W performance metrics, and average estimated system slowdown $\bar{\beta}$. The α parameter in Equation (5.5) was set to 1.0. The results show the higher the uncertainty (i.e., ASE) in slowdown data samples provided by upstream slowdown meters, the more difficult to take the right decision, thus reaching higher values of $\bar{\beta}$ and inferior rate of completed jobs E_J. In spite of this, the energy efficiency E_M and working efficiency E_W decrease as well, because less number of jobs are being completed.

5.5.2 Network-bound Workloads

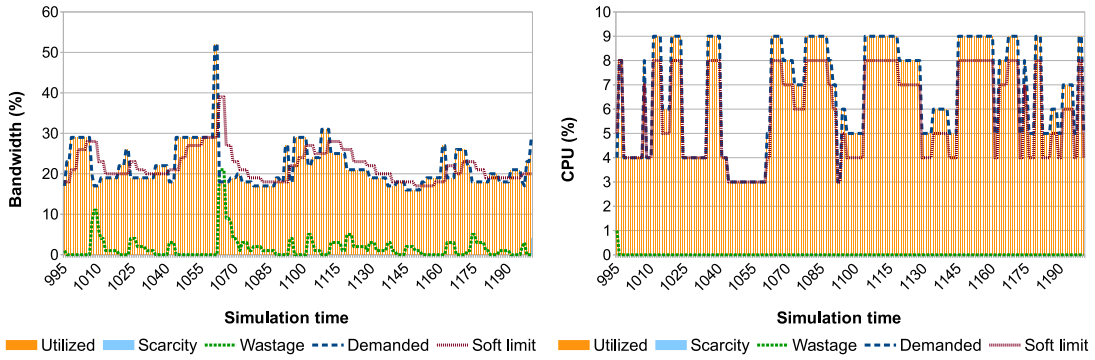
In this set of simulations, it is investigated how well KFIO + PIASA mechanism performs to fulfil network-bound application QoS requirements which workload demands change abruptly. Unlike Garg et al. [GTGB14] alternative, the KFIO + PIASA mechanism performs dynamic consolidation of workloads that combined may exceed the total available resources in the host. The cloud manager module invokes the PIASA scheduling algorithm to readjust the VMs to PMs mapping if the performance deviates at least 5% from the figure previously specified in the last schedule. The α parameter in Equation (5.5) was set to 0.50 to achieve a good compromise between performance and energy efficiency. The demand of resources oscillates with average value of 0%, and standard deviation of 15%. The results are shown in Table 5.4.

The results show that PIASA algorithm outperforms Garg et al. [GTGB14] strategy, providing enough resources to about 95.16% of applications so they can successfully serve at least 95% of requests. Since PIASA consumes additional resources to fulfil application QoS requirements, more energy "Energy (MJ)" is consumed.

In Figure 5.5 is drawn the temporal dynamic adjustment of bandwidth and CPU resources for a VM running a network-bound workload. The deviation threshold beyond which an adjustment was triggered was set to $\delta = 5\%$. The α parameter in Equation (5.5) was set to 0.50. Figure 5.5(a) plots the bandwidth usage during a fraction of the runtime of one VM. While the y-axis represents the fraction of server's bandwidth the VM consumes according to workload needs, the x-axis represents the simulation time (a time unit corresponds to 11 seconds). The bandwidth varies

Table 5.4: Performance of scheduling algorithms in the management of network-bound workloads that present oscillations in demand of resources (oscillations in requested bandwidth with average value of 0%, and standard deviation of 15%). The α parameter in Equation (5.5) was set to 0.50.

Algorithm	100%	[95,100[%	[90,95[%	[85,90[%	[0,85[%	Energy (MJ)
Garg et al. [GTGB14]	0.0%	3.2%	36.1%	30.0%	30.7%	262.1
PIASA	37.8%	57.4%	2.8%	1.5%	0.5%	431.5



(a) Bandwidth adapting.

(b) CPU adapting.

Figure 5.5: Dynamic adjustment of bandwidth and CPU resources for a VM running a network-bound workload. The deviation threshold beyond which an adjustment was triggered was set to $\delta = 5\%$. The α parameter in Equation (5.5) was set to 0.50.

along time, according to the number of requests per time unit and the size of file to download. The "Demanded" bandwidth accounts for the bandwidth required at each time unit. The "Utilized" bandwidth represents the bandwidth that the application was able to use during runtime, while the "Soft limit" expresses the minimum guaranteed bandwidth the algorithm decided to assign to the application. "Scarcity" is the difference between the resources demanded and utilized, and "Wastage" is the difference between the soft limit and demanded resources. The figure shows that the mechanism is responsive enough, dealing with performance deviations by expanding and shrinking resources assigned to the application. In spite of this, the "Scarcity" is always maintained at 0, and the "Wastage" line is below 5% in 95.6% of the time, and below 2% in 74.6% of the time, for a maximum oscillation of 36% in network I/O demand.

Figure 5.5(b) draws the CPU resource usage for the same fraction of the runtime of the VM hosting the web application. The y-axis plots the fraction of server's CPU utilized by the VM according to workload needs. The figure draws the exactly same metrics as in Figure 5.5(a). The maximum oscillation in demand of CPU resource was 6%, and the "Wastage" and "Scarcity" show that the mechanism was able to not miss nor to waste CPU capacity.

5.5.3 Mixture of CPU- and Network-bound Workloads

This section assesses the efficacy of KFLR (CPU-bound workloads) + KFIO (network-bound workloads) + PIASA to enforce the diverse QoS requirements of applications with heterogeneous workloads, while optimizing the energy efficiency of the whole system. The value of α in Equation (5.5) is varied in $\{1.0, 0.75, 0.50, 0.25, 0.0\}$. Because the level of consolidation is very low for $\alpha = 0.0$, the effects produced by the energy optimizing mechanism are irrelevant. In this sense, the consolidation mechanism is switched off for $\alpha = 0.0$. The deviation threshold beyond which an adjustment was triggered for network-intensive workloads was set to 5%. The slowdown data samples provided by the upstream slowdown meter have an average slowdown error of 10%, and standard deviation of 20%.

Table 5.5 shows the results. The $\bar{H}$ stands for the average heterogeneity degree (described in Section 5.3.2), measured for all servers along simulation. The results demonstrate the trade-off between consolidation of VMs to improve energy efficiency versus higher levels of fulfilment of application QoS needs, as α evolves. For $\alpha = 1.0$, the mechanism achieves the best results in terms of relationship between Mflops per Joule E_M, and work produced for average power consumption E_W, corresponding to the case of less energy consumption "Energy (MJ)". However, the amount of CPU-bound workloads completed E_J, as well the number of network-bound applications serving successfully more than 95% of requests E_R, is the lowest for all α, representing 95.3% and 86.9%, respectively. Two factors explain these results for E_J and E_R: (1) the consolidation level VMs/PM is the higher, causing more performance interference as well, since higher number of tasks share the node's resources; (2) the value of the average heterogeneity level $\bar{H}$ is the higher, meaning that more workloads of the same type are consolidated per node, thus increasing performance interference. Therefore, the system gets power-efficient biased as α gets 1.0. In turn, when the parameter α tends to 0, the number of VMs serving successfully more than 95% of the network requests E_R reaches 96.4%, and the completion rate of users jobs E_J increases to 100%. Several factors contributed to these results: (1) the consolidation of VMs, VMs/PM diminishes, leading to minor degree in performance interference among co-hosted workloads; (2) the number of migrations "VM Migs (%)" decreases, hence tending for a higher stability of the whole system; (3) $\bar{H}$ approaches to 0, meaning that there is extra care in hosting CPU- together with network-intensive workloads. However, since consolidation level is inferior, the energy consumption "Energy (MJ)" increases, which lowers E_W and E_M. The variable $\bar{H}$ gets negative in the five sets of experiments because the number of CPU-bound workloads is higher than the total number of network-bound workloads.

5.6 Conclusions

Cloud data centers apply virtualization because it offers fault isolation and improved manageability through dynamic resource provisioning and live migration of VMs. Consolidation of VMs is a very common technique to improve energy efficiency and reduce operational costs. However,

Table 5.5: Performance of proposed KFLR + KFIO + PIASA, while varying α in the management of applications. The deviation threshold beyond which an adjustment was triggered for network-intensive workloads was set to 5%. The α parameter in Equation (5.5) is varied from 1.00 to 0.00. Because the level of consolidation is very low for $\alpha = 0.0$, the consolidation mechanism was switched off in this case.

α	E_R (%)	E_J (%)	VMs/PM	VM Migs (%)	$\bar{H}$	Energy (MJ)	E_M (Mflops/J)	E_W (%)
1.00	86.9	95.3	9.14	28.7	-0.95	396.7	0.020	35.25
0.75	93.5	96.3	8.23	17.9	-0.93	433.2	0.019	34.53
0.50	95.6	98.0	7.26	12.2	-0.87	480.5	0.018	32.35
0.25	95.6	99.2	6.66	8.5	-0.80	512.5	0.017	29.42
0.00	96.4	100.0	4.64	1.0	-0.04	608.1	0.015	17.66

virtualization does not provide performance isolation, hence intensifying performance deviation among co-hosted VMs that contend for shared hardware resources. In the case of CPU-bound tasks, the sharing of on-chip resources can cause slowdown in their execution, and ultimately fail the deadlines. On the other hand, network-bound workloads are bursty, causing oscillation in demanding of resources, which can lead to violations of the SLA if available resources are less then demanded or assigned to.

This study proposed a performance and energy efficiency enforcing mechanism, composed of performance deviation estimators, KFLR for CPU-bound tasks and KFIO for network-bound applications, and a scheduling algorithm. The objective is to guarantee diverse application QoS requirements, while maximizing the energy efficiency to execute the workloads.

In order to analyse the performance of the proposed mechanism, an extensive set of simulations was carried out by injecting two types of workloads, namely CPU- and network-bound workloads. The workload characteristics of both sets follow the latest version of the Google cloud tracelogs. The first set of simulations analysed the efficacy of the mechanism in scheduling CPU-bound workloads, without and with slowdown caused by interference among co-hosted VMs. The results showed that proposed algorithm PIASA outperformed a state-of-the-art alternative, completing almost 96% of the submitted jobs. In the absence of slowdown, PIASA showed to be able to produce, at least, the same amount of work as alternative state-of-the-art algorithm, but consuming less energy. In the third set of simulations, the mechanism showed to be capable of assuring that almost 96% of network-bound applications could serve successfully more than 95% of requests. Finally, the proposed mechanism was tested with a mixture of CPU- and network-bound workloads. Considering that in real-world environments the SLA contract terms allow a mean performance degradation of 1–5%, the proposed performance enforcing mechanism can be applied effectively to fulfil expected QoS of applications and reduce energy costs up to 21%.

Chapter 6

Conclusions and Future Directions

This chapter summarizes the research work on resource management in cloud data centers, aiming at optimization of energy efficiency in order to reduce operational costs. At the same time, the work considered failures in servers and performance interference among co-hosted tasks with diverse QoS requirements, since these are important factors in current cloud data centers because they directly constrained the QoS provided to users' applications. The main findings are highlighted and future research problems related with the area are discussed.

6.1 Findings and Contributions

CLOUD computing providers are under great pressure to reduce operational costs and increase the return of investment. Energy consumption in data centers represent a very important fraction of the whole cost of operation. Consolidation of VM instances on a smaller number of physical servers improve the utilization of resources. In this sense, idle physical servers can be turned off or operated in low power mode, thus eliminating idle power consumption and costs associated with energy consumption. Because failures also contribute to increased energy consumption, since crash failures result in a loss of work and energy used, dynamic consolidation of VMs must be carried out by considering the reliability of physical servers. Even though VM consolidation contributes to improve energy efficiency in the data center, interference among VMs in a physical server may degrade the performance of applications running on these VMs.

This thesis has proposed a suite of improved mechanisms for implementing dynamic VM consolidation in IaaS clouds. The proposed approach improves the utilization of data center resources and reduces energy consumption, while satisfying the defined QoS requirements of applications. However, fulfilling the diversity of QoS requirements of different types of workloads imply extra requirements, such as implementing failure tolerance and performance enforcing techniques. To tackle the formulated research challenges in the management of IaaS clouds, this thesis delineates each of the objectives in Chapter 1. To better understand the advantages and limitations of virtualization, Chapter 2 presents an in-depth review of virtualization technology. In the same direction,

Chapter 3 addressed a state-of-the-art review in the area of energy-efficient and failure tolerance resource management in computing systems.

In Chapter 4, it is presented a multi-objective approach that dynamically creates and manages energy-efficient and high-available distributed VM infrastructures for networked computing systems. The Chapter delivers an approach that combines an energy optimizing mechanism to detect and mitigate energy inefficiencies, and two improved scheduling algorithms, POFAME and POFARE, that execute power- and failure-aware decisions to enable the execution of users' jobs with a high level of SLA accomplishment. By applying proactive fault tolerance to address node failures and consolidation to optimize energy efficiency, POFARE and POFAME are able to simultaneously increase the amount of useful Mflops processed per energy unit, as well as the number of jobs completed by the consumed power, in cases where the infrastructure nodes are subject to failure. The analysis to the performance of the proposed algorithms comprised a set of simulations and real platform evaluation, by injecting random synthetic jobs and jobs using the latest version of the Google cloud tracelogs. POFARE and POFAME apply a different strategy in the assignment of resources to VMs, in the sense that the former sets up the cap parameter from the Xen credit scheduler to execute tasks with the minimum required resources. The results have shown that POFARE is the algorithm that performs better for different values of failure prediction accuracy and diverse average task length to MTBF ratios, delivering more work for less energy consumed. A common best-fit strategy can deliver better results for cases of failure prediction accuracy inferior than 20%.

Virtualization is the crucial technology in current data centers, because of its capacity to offer fault isolation and improved manageability through dynamic resource provisioning and live migration of VMs. However, virtualization does not provide performance isolation, which intensifies performance deviation among co-hosted VMs contending for shared hardware resources. Besides that, in the case of network-bound workloads, oscillations in demanding of resources due to their typical bursty behaviour, leads to violations of the SLA if available resources are less then demanded or assigned to. To address this problem, in Chapter 5 is proposed a performance and energy efficiency enforcing mechanism, composed of two performance deviation estimators, KFLR for CPU-bound tasks and KFIO for network-bound applications, and PIASA - an improved scheduling algorithm. An extensive set of simulations was carried out by injecting two types of applications, CPU- and network-bound applications, which characteristics follow the latest version of the Google cloud tracelogs. The results have shown that the proposed mechanism is able to produce, at least, the same amount of work as alternative state-of-the-art algorithms, but consuming less energy. In the presence of performance deviations, the proposed mechanism has shown to be capable of fulfilling the different QoS requirements of two different types of workloads. Moreover, considering that in real-world environments the SLA contract terms allow a mean performance degradation of 1–5%, the proposed performance enforcing mechanism can be applied effectively to fulfil expected QoS of applications and reduce energy costs up to 21%.

6.2 Future Research Directions

Resource provisioning in the cloud has advanced significantly in recent years. This thesis has contributed with substantial improvements in this field, by proposing alternatives to optimize energy efficiency, achieve fault tolerance, and meet application QoS requirements. Nevertheless, there are a number of open research challenges that need to be addressed and can serve as a starting point for future research.

6.2.1 Other Sources of Energy Consumption

Energy consumption in data centers is mainly dictated by servers, and in particular by the CPU component [BH09]. Due to continuous improvements of the CPU power efficiency (e.g., DVFS technique), the CPU no longer dominates power consumption by a server. In today's servers, memory deployed is growing in an attempt to follow the increment of CPU power per server as it may be the bottleneck resources for certain applications. In this regarding, the power consumed by memory starts representing an important fraction of the total power drawn in a server. Merkel and Bellosa [MB08] had already contributed with some work in this area.

Because some energy is always lost as heat (e.g., due to inefficient power supplies, unnecessary processes, etc.), considerable energy is consumed in data centers for cooling the space. Thermal management in data centers is essential to mitigate hotspots and to keep temperature within a safe operating range. Cooling loads can be reduced by several ways, such as using more efficient servers and power supplies, consolidation into hot and cold aisles. Therefore, thermal-aware scheduling of applications can be used to preserve safe temperature of the resources. Xu and Fortes [XF11] had already proposed some work in this field.

Communication is one of the largest consumers of energy. Surprisingly enough, only very recently network infrastructures have received attention, despite the existing detailed studies about energy-saving routing protocols in wireless sensor networks because of the specific issues of batteries. One must pay more attention to the energy consumption of the transmission and switching networks that are the mean to connecting users to the cloud. Some research in this field includes [Mah11].

6.2.2 Combination with Fault-tolerant Reactive Schemes

Failures contribute for increasing energy consumption and creating bad reputation about the services' dependability perceived by end-users. Despite the advantages of proactive fault tolerance schemes in reducing the overhead to create reliable and high-available systems, such schemes rely on the capacity of predicting failure. However, dependability at all cost is not a solution, because it ends up increasing the energy consumption and operational costs. To this end, it is necessary to investigate the articulation between proactive and reactive (e.g., based on checkpointing) fault-tolerant schemes and properly analyse the dependability vs. energy trade-off. Fault tolerance as a service may be issued, as initially addressed by Jhawar et al. [JPS13].

6.2.3 Performance Interference and Application Affinities

One of the works in this thesis considered workloads with dissimilar resources type needs to manage IaaS cloud environments and reduce performance interference. This problem assumes special relevancy on modern chip multiprocessors (CMP), where architectures consist of multiple processing cores on a single die and cache and memory are shared by all cores. Since CMP cores are not fully independent processors, the performance degradation due to competition for shared resources can be even worse. In this regarding, time and space-sharing properties in scheduling algorithms assume a very important role, further highlighting the importance of data locality. To this end, clustering applications with data affinity and thread-level scheduling have been shown to be very effective at helping mitigating the performance losses. Further research in the field of contention-aware scheduling and data locality analysis needs to be addressed with the near ubiquitous proliferation of CMP. Some research in this area includes [JMJ06, JZTS10, ZSB+12].

References

[AB14] H. Arabnejad and J.G. Barbosa. List scheduling algorithm for heterogeneous systems by an optimistic cost table. *Parallel and Distributed Systems, IEEE Transactions on*, 25:682–694, March 2014.

[AD09] Django Armstrong and Karim Djemame. Towards quality of service in the cloud. In *Proc. of the 25th UK Performance Engineering Workshop*, 2009.

[Ada66] Robin J Adair. *A virtual machine system for the 360/40*. International Business Machines Corporation, Cambridge Scientific Center, 1966.

[AFG+10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[ALRL04] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, 2004.

[ANE12] Saeid Abrishami, Mahmoud Naghibzadeh, and Dick HJ Epema. Cost-driven scheduling of grid workflows using partial critical paths. *Parallel and Distributed Systems, IEEE Transactions on*, 23(8):1400–1414, 2012.

[B+08] Richard Brown et al. Report to congress on server and data center energy efficiency: Public law 109-431. *Lawrence Berkeley National Laboratory*, 2008.

[BAB12] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future generation computer systems*, 28:755–768, 2012.

[BAHT07] Jayant Baliga, Robert WA Ayre, Kerry Hinton, and RodneyS Tucker. Efficient power supplies for data center and enterprise servers. *Tech. Rep.*, ET 07.01, 2007.

[BAHT11] Jayant Baliga, Robert WA Ayre, Kerry Hinton, and RodneyS Tucker. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167, 2011.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[BGDG+10] Andreas Berl, Erol Gelenbe, Marco Di Girolamo, Giovanni Giuliani, Hermann De Meer, Minh Quan Dang, and Kostas Pentikousis. Energy-efficient cloud computing. *The Computer Journal*, 53(7):1045–1051, 2010.

[BGJ06] Vandy Berten, Joël Goossens, and Emmanuel Jeannot. On the distribution of sequential jobs in random brokering for heterogeneous computational grids. *Parallel and Distributed Systems, IEEE Transactions on*, 17(2):113–124, 2006.

[BH07] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *IEEE computer*, 40(12):33–37, 2007.

[BH09] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. *Synthesis lectures on computer architecture*, 4(1):1–108, 2009.

[BJ03] Francis R Bach and Michael I Jordan. Kernel independent component analysis. *The Journal of Machine Learning Research*, 3:1–48, 2003.

[BSS14] Alexandru Butoi, Alexandru Stan, and Gheorghe Cosmin Silaghi. Autonomous management of virtual machine failures in iaas using fault tree analysis. In *Economics of Grids, Clouds, Systems, and Services*, pages 206–221. Springer, 2014.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[BZF10] Sergey Blagodurov, Sergey Zhuravlev, and Alexandra Fedorova. Contention-aware scheduling on multicore systems. *ACM Transactions on Computer Systems (TOCS)*, 28(4):8, 2010.

[C+10] European Commission et al. Code of conduct on data centres energy efficiency: Version 1.0, 2010.

[CFH+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[CGV07] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[CH11] Ron C Chiang and H Howie Huang. Tracon: Interference-aware scheduling for data-intensive applications in virtualized environments. In *proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, page 47. ACM, 2011.

[CK11] Jinkyun Cho and Byungseon Sean Kim. Evaluation of air management system's thermal performance for superior cooling efficiency in high-density data centers. *Energy and buildings*, 43(9):2145–2155, 2011.

[CKK11] Giuliano Casale, Stephan Kraft, and Diwakar Krishnamurthy. A model of storage i/o performance interference in virtualized systems. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 34–39. IEEE, 2011.

[CLM+08] Brendan Cully, Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, and Andrew Warfield. Remus: High availability via asynchronous virtual machine replication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 161–174. San Francisco, 2008.

[Cou10] National Research Council. *The Rise of Games and High Performance Computing for Modeling and Simulation*. The National Academies Press, Washington, DC, 2010.

[CSW+08] David Carrera, Malgorzata Steinder, Ian Whalley, Jordi Torres, and Eduard Ayguadé. Enabling resource sharing between transactional and batch workloads using dynamic application placement. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, pages 203–222. Springer-Verlag New York, Inc., 2008.

[Dab09] Christopher Dabrowski. Reliability in grid computing systems. *Concurrency and Computation: Practice and Experience*, 21(8):927–959, 2009.

[DEAS11] Javier Delgado, Anas Salah Eddin, Malek Adjouadi, and Seyed Masoud Sadjadi. Paravirtualization for scientific computing: Performance analysis and prediction. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*, pages 536–543. IEEE, 2011.

[DFB+12] Tyler Dwyer, Alexandra Fedorova, Sergey Blagodurov, Mark Roth, Fabien Gaud, and Jian Pei. A practical method for estimating performance degradation on multicore processors, and its application to hpc workloads. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 83. IEEE Computer Society Press, 2012.

[DGF+91] Jean-Louis Deneubourg, Simon Goss, Nigel Franks, Ana Sendova-Franks, Claire Detrain, and Laeticia Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 356–363, 1991.

[DL11] Jack Dongarra and Piotr Luszczek. Linpack benchmark. In *Encyclopedia of Parallel Computing*, pages 1033–1036. Springer, 2011.

[DMR09] Gaurav Dhiman, Giacomo Marchetti, and Tajana Rosing. vgreen: a system for energy efficient computing in virtualized environments. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 243–248. ACM, 2009.

[DQLW15] Youwei Ding, Xiaolin Qin, Liang Liu, and Taochun Wang. Energy efficient scheduling of virtual machines in cloud with deadline constraint. *Future Generation Computer Systems*, 50:62–74, 2015.

[DSL+08] Ewa Deelman, Gurmeet Singh, Miron Livny, Bruce Berriman, and John Good. The cost of doing science on the cloud: the montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 50. IEEE Press, 2008.

[ELSC13] Ifeanyi P Egwutuoha, David Levy, Bran Selic, and Shiping Chen. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. *The Journal of Supercomputing*, 65(3):1302–1326, 2013.

[Erl05] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[EVNS09] Christian Engelmann, Geoffroy R Vallee, Thomas Naughton, and Stephen L Scott. Proactive fault tolerance using preemptive migration. In *Parallel, Distributed and Network-based Processing, 2009 17th Euromicro International Conference on*, pages 252–257. IEEE, 2009.

[EYD07] Deniz Ersoz, Mazin S Yousif, and Chita R Das. Characterizing network traffic in a cluster-based, multi-tier data center. In *Distributed Computing Systems, 2007. ICDCS'07. 27th International Conference on*, pages 59–59. IEEE, 2007.

[FD92] Emanuel Falkenauer and Alain Delchambre. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 1186–1192. IEEE, 1992.

[Fu10] Song Fu. Failure-aware resource management for high-availability computing clusters with distributed virtual machines. *Journal of Parallel and Distributed Computing*, 70(4):384–393, 2010.

[FWB07] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 13–23. ACM, 2007.

[FX07a] Song Fu and Cheng-Zhong Xu. Exploring event correlation for failure prediction in coalitions of clusters. In *Supercomputing, 2007. SC'07. Proceedings of the 2007 ACM/IEEE Conference on*, pages 1–12. IEEE, 2007.

[FX07b] Song Fu and Cheng-Zhong Xu. Quantifying temporal and spatial correlation of failure events for proactive management. In *Reliable Distributed Systems, 2007. SRDS 2007. 26th IEEE International Symposium on*, pages 175–184. IEEE, 2007.

[GHMP08] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM Computer Communication Review*, 39(1):68–73, 2008.

[GLKS11] Sriram Govindan, Jie Liu, Aman Kansal, and Anand Sivasubramaniam. Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 22. ACM, 2011.

[GMDC+13] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. Energy-aware simulation with dvfs. *Simulation Modelling Practice and Theory*, 39:76–91, 2013.

[Gol74] Robert P Goldberg. Survey of virtual machine research. *Computer*, 7(6):34–45, 1974.

[Goo11] Google. Google cluster data v2. http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1, 2011. Accessed on 25/03/2013.

[GTGB14] Saurabh Kumar Garg, Adel Nadjaran Toosi, Srinivasa K Gopalaiyengar, and Rajkumar Buyya. Sla-based virtual machine management for heterogeneous workloads in a cloud datacenter. *Journal of Network and Computer Applications*, 45:108–120, 2014.

[GTX13] Peter Garraghan, Paul Townend, and Jie Xu. An analysis of the server characteristics and resource utilization in google cloud. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 124–131. IEEE, 2013.

[HA12] Y. Hashimoto and K. Aida. Evaluation of performance degradation in hpc applications with vm consolidation. In *Networking and Computing (ICNC), 2012 Third International Conference on*, pages 273–277, Dec 2012.

[Har90] Andrew C Harvey. *Forecasting, structural time series models and the Kalman filter*. Cambridge university press, 1990.

[HF05] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1. IEEE Computer Society, 2005.

[HG09] Michael R Hines and Kartik Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60. ACM, 2009.

[HGM+02] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, P Schroeder, Jasper Spaans, and Pedro Larroy. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, page 213, 2002.

[HL13] Qun Huang and Patrick PC Lee. An experimental study of cascading performance interference in a virtualized environment. *ACM SIGMETRICS Performance Evaluation Review*, 40(4):43–52, 2013.

[HLM06] Fabien Hermenier, Nicolas Loriant, and Jean-Marc Menaud. Power management in grid computing with xen. In *Frontiers of High Performance Computing and Networking–ISPA 2006 Workshops*, pages 407–416. Springer, 2006.

[HUM+11] Kai-Yuan Hou, Mustafa Uysal, Arif Merchant, Kang G Shin, and Sharad Singhal. Hydravm: Low-cost, transparent high availability for virtual machines. Technical report, HP Laboratories, Tech. Rep, 2011.

[Jac12] Kevin Jackson. *OpenStack cloud computing cookbook*. Packt Publishing Ltd, 2012.

[JCC+10] Daeyong Jung, SungHo Chin, KwangSik Chung, Taeweon Suh, HeonChang Yu, and JoonMin Gil. An effective job replication technique based on reliability and performance in mobile grids. In *Advances in Grid and Pervasive Computing*, pages 47–58. Springer, 2010.

[JD10] Gideon Juve and Ewa Deelman. Scientific workflows and clouds. *Crossroads*, 16(3):14–18, 2010.

[JMJ06] Aamer Jaleel, Matthew Mattina, and Bruce Jacob. Last level cache (llc) performance of data mining workloads on a cmp-a case study of parallel bioinformatics workloads. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 88–98. IEEE, 2006.

[Joh67] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[JPS13] Ravi Jhawar, Vincenzo Piuri, and Marco Santambrogio. Fault tolerance management in cloud computing: A system-level perspective. *Systems Journal, IEEE*, 7(2):288–297, 2013.

[JZTS10] Yunlian Jiang, Eddy Z Zhang, Kai Tian, and Xipeng Shen. Is reuse distance applicable to data locality analysis on chip multiprocessors? In *Compiler Construction*, pages 264–282. Springer, 2010.

[KBB09] Kyong Hoon Kim, Anton Beloglazov, and Rajkumar Buyya. Power-aware provisioning of cloud resources for real-time services. In *Proceedings of the 7th International Workshop on Middleware for Grids, Clouds and e-Science*, page 1. ACM, 2009.

[Kec02] Dimitri Kececioglu. *Reliability engineering handbook*, volume 1. DEStech Publications, Inc, 2002.

[KEY12] Shin-gyu Kim, Hyeonsang Eom, and Heon Y Yeom. Virtual machine scheduling for multicores considering effects of shared on-chip last level cache interference. In *Green Computing Conference (IGCC), 2012 International*, pages 1–6. IEEE, 2012.

[KFK08] James M Kaplan, William Forrest, and Noah Kindler. Revolutionizing data center energy efficiency. *McKinsey & Company, Tech. Rep*, 2008.

[KGB13] Atefeh Khosravi, Saurabh Kumar Garg, and Rajkumar Buyya. Energy and carbon-efficient placement of virtual machines in distributed cloud data centers. In *Euro-Par 2013 Parallel Processing*, pages 317–328. Springer, 2013.

[KJIE10] Derrick Kondo, Bahman Javadi, Alexandru Iosup, and Dick Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 398–407. IEEE, 2010.

[KK10] Israel Koren and C. Mani Krishna. *Fault-tolerant systems*. Morgan Kaufmann, 2010.

[KKB+07] Younggyun Koh, Rob Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu. An analysis of performance interference effects in virtual environments. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 200–209. IEEE, 2007.

[KL03] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[KMB+11] Matthias Keller, Dirk Meister, André Brinkmann, Christian Terboven, and Christian Bischof. escience cloud infrastructure. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 188–195. IEEE, 2011.

[KMHK12] Melanie Kambadur, Tipp Moseley, Rick Hank, and Martha A Kim. Measuring interference between live datacenter applications. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 51. IEEE Computer Society Press, 2012.

[Koo07] Jonathan G Koomey. Estimating total power consumption by servers in the us and the world, 2007.

[Koo11] Jonathan Koomey. Growth in data center electricity use 2005 to 2010, 2011.

[KRA12] Jungsoo Kim, Martino Ruggiero, and David Atienza. Free cooling-aware dynamic power management for green datacenters. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 140–146. IEEE, 2012.

[L+13] Bart Lannoo et al. Overview of ict energy consumption. *Deliverable D8*, 1, 2013.

[LC12] Zitao Liu and Sangyeun Cho. Characterizing machines and workloads on a google cluster. In *Parallel Processing Workshops (ICPPW), 2012 41st International Conference on*, pages 397–403. IEEE, 2012.

[LCWS+09] Horacio Andrés Lagar-Cavilla, Joseph Andrew Whitney, Adin Matthew Scannell, Philip Patchin, Stephen M Rumble, Eyal De Lara, Michael Brudno, and Mahadev Satyanarayanan. Snowflock: rapid virtual machine cloning for cloud computing. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 1–12. ACM, 2009.

[LDL+08] Scott Loveland, Eli M Dow, Frank LeFevre, Duane Beyer, and Phil F Chan. Leveraging virtualization to optimize high-availability system configurations. *IBM Systems Journal*, 47(4):591–604, 2008.

[Len15] Ricardo Lent. Analysis of an energy proportional data center. *Ad Hoc Networks*, 25:554–564, 2015.

[LHK+12] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, Galen M Shipman, and Chita R Das. D-factor: a quantitative model of application slow-down in multi-resource shared systems. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):271–282, 2012.

[LHP15] Li Lei, An Huiyao, and Zhang Peng. Study on last-level cache management strategy of the chip multi-processor. *Appl. Math*, 9:661–670, 2015.

[LL06] Yawei Li and Zhiling Lan. Exploit failure prediction for adaptive fault-tolerance in cluster computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 8–pp. IEEE, 2006.

[LO10] Laurent Lefèvre and Anne-Cécile Orgerie. Designing and evaluating an energy efficient cloud. *The Journal of Supercomputing*, 51(3):352–373, 2010.

[LZ12] Young Choon Lee and Albert Y Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[M+98] Gordon E Moore et al. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, 1998.

[MA04] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[Mah11] Toktam Mahmoodi. Energy-aware routing in the cognitive packet network. *Performance Evaluation*, 68:338–346, 2011.

[MB08] Andreas Merkel and Frank Bellosa. Memory-aware scheduling for energy efficiency on multicore processors. *HotPower*, 8:123–130, 2008.

[mbls10] Hardware monitoring by lm sensors. Lm_sensors - linux hardware monitoring. http://www.lm-sensors.org/, 2010. Accessed on 24/12/2014.

[MBS13] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Adaptive resource configuration for cloud infrastructure management. *Future Generation Computer Systems*, 29(2):472–487, 2013.

[MCC04] Matthew L Massie, Brent N Chun, and David E Culler. The ganglia distributed monitoring system: design, implementation, and experience. *Parallel Computing*, 30(7):817–840, 2004.

[MCTL+14] Ali José Mashtizadeh, Min Cai, Gabriel Tarasuk-Levin, Ricardo Koller, Tal Garfinkel, and Sreekanth Setty. Xvmotion: unified virtual machine migration over long distance. In *Proceedings of the 2014 USENIX conference on USENIX Annual Technical Conference*, pages 97–108. USENIX Association, 2014.

[MDD10] Michele Mazzucco, Dmytro Dyachuk, and Ralph Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 131–138. IEEE, 2010.

[MGTX13] Ismael Solis Moreno, Peter Garraghan, Paul Townend, and Jie Xu. An approach for characterizing workloads in google cloud to derive realistic resource utilization models. In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pages 49–60. IEEE, 2013.

[MGTX14] I.S. Moreno, P. Garraghan, P. Townend, and Jie Xu. Analysis, modeling and simulation of workload patterns in a large-scale utility cloud. *Cloud Computing, IEEE Transactions on*, 2:208–221, April 2014.

[MGW09] David Meisner, Brian T Gold, and Thomas F Wenisch. Powernap: eliminating server idle power. In *ACM Sigplan Notices*, volume 44, pages 205–216. ACM, 2009.

[Min06] Corey Minyard. A gentle introduction with openipmi. *Montavista Software*, 2006.

[MLP+13] Yiduo Mei, Ling Liu, Xing Pu, Sankaran Sivathanu, and Xiaoshe Dong. Performance analysis of network i/o workloads in virtualized data centers. *Services Computing, IEEE Transactions on*, 6:48–63, 2013.

[MLPS10] Yiduo Mei, Ling Liu, Xing Pu, and Sankaran Sivathanu. Performance measurements and analysis of network i/o applications in virtualized cloud. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 59–66. IEEE, 2010.

[MMP13] Carlo Mastroianni, Michela Meo, and Giuseppe Papuzzo. Probabilistic consolidation of virtual machines in self-organizing cloud data centers. *Cloud Computing, IEEE Transactions on*, 1(2):215–228, 2013.

[MN06] James W Mickens and Brian D Noble. Exploiting availability prediction in distributed systems. *Ann Arbor*, 1001:48103, 2006.

[MTS+12] Jason Mars, Lingjia Tang, Kevin Skadron, Mary Lou Soffa, and Robert Hundt. Increasing utilization in modern warehouse-scale computers using bubble-up. *Micro, IEEE*, 32(3):88–99, 2012.

[MYXW13] Ismael Solis Moreno, Renyu Yang, Jie Xu, and Tianyu Wo. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, pages 1–8. IEEE, 2013.

[NBF+10] Oliver Niehorster, Andre Brinkmann, Gregor Fels, Jens Kruger, and Jens Simon. Enforcing slas in scientific clouds. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 178–187. IEEE, 2010.

[NESS09] Ripal Nathuji, Paul England, Parag Sharma, and Abhishek Singh. Feedback driven qos-aware power budgeting for virtualized servers. In *Fourth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID)*, 2009.

[NKG10] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah. Q-clouds: managing performance interference effects for qos-aware clouds. In *Proceedings of the 5th European conference on Computer systems*, pages 237–250. ACM, 2010.

[NM65] John A Nelder and Roger Mead. A simplex method for function minimization. *Computer journal*, 7(4):308–313, 1965.

[NMES07] Arun Babu Nagarajan, Frank Mueller, Christian Engelmann, and Stephen L Scott. Proactive fault tolerance for hpc with xen virtualization. In *Proceedings of the 21st annual international conference on Supercomputing*, pages 23–32. ACM, 2007.

[NPI+08] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI*, volume 8, pages 323–336, 2008.

[OSM+04] Adam J Oliner, Ramendra K Sahoo, José E Moreira, Manish Gupta, and Anand Sivasubramaniam. Fault-aware job scheduling for bluegene/l systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 64. IEEE, 2004.

[PGMAdM11] F Pontes Guimaraes and Alba Cristina Magalhaes Alves de Melo. User-defined adaptive fault-tolerant execution of workflows in the grid. In *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, pages 356–362. IEEE, 2011.

[PGP10] Ehsan Pakbaznia, Mohammad Ghasemazar, and Massoud Pedram. Temperature-aware dynamic resource provisioning in a power-optimized datacenter. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 124–129. European Design and Automation Association, 2010.

[Phi05] Ian Philp. Software failures and the road to a petaflop machine. In *HPCRI: 1st Workshop on High Performance Computing Reliability Issues, in Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA-11)*, 2005.

[PLM+10] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. Understanding performance interference of i/o workload in virtualized cloud environments. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 51–58. IEEE, 2010.

[PLM+13] Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, Calton Pu, and Yuanda Cao. Who is your neighbor: Net i/o performance interference in virtualized clouds. *Services Computing, IEEE Transactions on*, 6(3):314–329, 2013.

[PPF09] Kassian Plankensteiner, Radu Prodan, and Thomas Fahringer. A new fault tolerance heuristic for scientific workflows in highly distributed environments based on resubmission impact. In *e-Science, 2009. e-Science'09. Fifth IEEE International Conference on*, pages 313–320. IEEE, 2009.

[Pro11] Open Compute Project. Energy efficiency. http://www.opencompute.org/about/energy-efficiency/, 2011. Accessed on 29/01/2015.

[PRS09] Pankesh Patel, Ajith H Ranabahu, and Amit P Sheth. Service level agreement in cloud computing. *The Ohio Center of Excellence in Knowledge-Enabled Computing*, 2009.

[PTS11] Andreas Polze, Peter Troger, and Felix Salfner. Timely virtual machine migration for pro-active fault tolerance. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 234–243. IEEE, 2011.

[RCA11] Purdue University RCAC. Carter cluster. https://www.rcac.purdue.edu/userinfo/resources/carter/, 2011. Accessed on 25/03/2014.

[RJQ+10] Ivan Rodero, Juan Jaramillo, Andres Quiroz, Manish Parashar, Francesc Guim, and Stephen Poole. Energy-efficient application-aware online provisioning for virtualized clouds and data centers. In *Green Computing Conference, 2010 International*, pages 31–45. IEEE, 2010.

[Rok10] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[RRK08] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. *HotPower*, 8:3–3, 2008.

[RTG+12] Charles Reiss, Alexey Tumanov, Gregory R Ganger, Randy H Katz, and Michael A Kozuch. Towards understanding heterogeneous clouds at scale: Google trace analysis. *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, 2012.

[RWGX13] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. Qos guarantees and service differentiation for dynamic cloud applications. *Network and Service Management, IEEE Transactions on*, 10(1):43–55, 2013.

[SB13a] Altino Sampaio and Jorge G Barbosa. Last-level cache interference-aware scheduling in scientific clouds. *Parallel & Cloud Computing*, 2:116–125, 2013.

[SB13b] Altino M Sampaio and Jorge G Barbosa. Dynamic power-and failure-aware cloud resources allocation for sets of independent tasks. In *Cloud Engineering (IC2E), 2013 IEEE International Conference on*, pages 1–10. IEEE, 2013.

[SB13c] Altino M Sampaio and Jorge G Barbosa. Optimizing energy-efficiency in high-available scientific cloud environments. In *Cloud and Green Computing (CGC), 2013 Third International Conference on*, pages 76–83. IEEE, 2013.

[SB14a] Altino M Sampaio and Jorge G Barbosa. Estimating effective slowdown of tasks in energy-aware clouds. In *Parallel and Distributed Processing with Applications (ISPA), 2014 IEEE International Symposium on*, pages 101–108. IEEE, 2014.

[SB14b] Altino M Sampaio and Jorge G Barbosa. A performance enforcing mechanism for energy- and failure-aware cloud systems. In *Green Computing Conference (IGCC), 2014 IEEE International*. IEEE, 2014.

[SB14c] Altino M Sampaio and Jorge G Barbosa. Towards high-available and energy-efficient virtual computing environments in the cloud. *Future Generation Computer Systems*, 40:30–43, 2014.

[SC08] David Schanzenbach and Henri Casanova. Accuracy and responsiveness of cpu sharing using xen's cap values. *University of Hawai 'i at Manoa Department of Information and Computer Sciences, Tech. Rep. ICS2008-05-01*, 2008.

[SC09] Gaurav Somani and Sanjay Chaudhary. Application performance isolation in virtualization. In *Cloud Computing, 2009. CLOUD'09. IEEE International Conference on*, pages 41–48. IEEE, 2009.

[SC10] Gaurav Somani and Sanjay Chaudhary. Performance isolation and scheduler behavior. In *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, pages 272–277. IEEE, 2010.

[SG07] Bianca Schroeder and Garth A Gibson. Understanding failures in petascale computers. In *Journal of Physics: Conference Series*, volume 78, page 012022. IOP Publishing, 2007.

[SG10] Bianca Schroeder and Garth A Gibson. A large-scale study of failures in high-performance computing systems. *Dependable and Secure Computing, IEEE Transactions on*, 7(4):337–350, 2010.

[Sho02] Martin L. Shooman. *Reliability of Computer Systems and Networks: Fault Tolerance,Analysis,and Design*. John Wiley & Sons, Inc., New York, NY, USA, 2002.

[SHRE13] Mina Sedaghat, Francisco Hernandez-Rodriguez, and Erik Elmroth. A virtual machine re-packing approach to the horizontal vs. vertical elasticity trade-off for cloud autoscaling. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, page 6. ACM, 2013.

[SJNL10] Lutz Schubert, Keith G Jeffery, and Burkard Neidecker-Lutz. *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010:–expert Group Report*. European Commission, Information Society and Media, 2010.

[SL12] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 936. John Wiley & Sons, 2012.

[SLM10] Felix Salfner, Maren Lenk, and Miroslaw Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010.

[SPE14] SPECpower. First quarter 2014 specpower_ssj2008 results. http://www.spec.org/power_ssj2008/results/res2014q1/, 2014. Accessed on 15/01/2015.

[SS09] Vladimir Stantchev and Christian Schröpfer. Negotiating and enforcing qos and slas in grid and cloud computing. In *Advances in grid and pervasive computing*, pages 25–35. Springer, 2009.

[SSF+10] Hossein Salami, Hamid Saadatfar, Farhad Rahmani Fard, S Kazem Shekofteh, and Hossein Deldari. Improving cluster computing performance based on job futurity prediction. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 6, pages V6–303. IEEE, 2010.

[SSGW11] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

[SVC12] Mark Stillwell, Frederic Vivien, and Henri Casanova. Dynamic fractional resource scheduling versus batch scheduling. *Parallel and Distributed Systems, IEEE Transactions on*, 23(3):521–529, 2012.

[TVS02] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems*, volume 2. Prentice Hall, 2002.

[TXM14] Paul Townend, Jie Xu, and Ismael Solis Moreno. An analysis of failure-related energy waste in a large-scale cloud environment. *IEEE Transactions on Emerging Topics in Computing*, page 1, 2014.

[VAN08a] Akshat Verma, Puneet Ahuja, and Anindya Neogi. pmapper: power and migration cost aware application placement in virtualized systems. In *Middleware 2008*, pages 243–264. Springer, 2008.

[VAN08b] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 175–184. ACM, 2008.

[VLWYH09] Gregor Von Laszewski, Lizhe Wang, Andrew J Younge, and Xi He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*, pages 1–10. IEEE, 2009.

[VMw07] VMware. Understanding full virtualization, paravirtualization, and hardware assist. http://www.vmware.com/files/pdf/VMware_paravirtualization.pdf, November 2007. Accessed on 10/07/2013.

[VNE+08] Geoffroy Vallee, Thomas Naughton, Christian Engelmann, Hong Ong, and Stephen L Scott. System-level virtualization for high performance computing. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 636–643. IEEE, 2008.

[VPB09] Christian Vecchiola, Suraj Pandey, and Rajkumar Buyya. High-performance cloud computing: A view of scientific applications. In *Pervasive Systems, Algorithms, and Networks (ISPAN), 2009 10th International Symposium on*, pages 4–16. IEEE, 2009.

[XF10] Jing Xu and Jose AB Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *Green Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 179–188. IEEE, 2010.

[XF11] Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 225–234. ACM, 2011.

[XSWJ08] Xianghua Xu, Peipei Shan, Jian Wan, and Yucheng Jiang. Performance evaluation of the cpu scheduler in xen. In *Information Science and Engineering, 2008. ISISE'08. International Symposium on*, volume 2, pages 68–72. IEEE, 2008.

[YAB+86] Michael Young, Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, and Avadis Tevanian. Mach: A new kernel foundation for unix development. In *Proceedings of the 1986 Summer USENIX Conference*, pages 93–113, 1986.

[YGK+10] Nezih Yigitbasi, Matthieu Gallet, Derrick Kondo, Alexandru Iosup, and Dick Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 65–72. IEEE, 2010.

[YS13] Iman I Yusuf and Heinz W Schmidt. Parameterised architectural patterns for providing cloud service fault tolerance with accurate costings. In *Proceedings of the 16th International ACM Sigsoft symposium on Component-based software engineering*, pages 121–130. ACM, 2013.

[YSP09] Iman I Yusuf, Heinz W Schmidt, and Ian D Peake. Evaluating recovery aware components for grid reliability. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 277–280. ACM, 2009.

[YSP11] Iman I Yusuf, Heinz W Schmidt, and Ian D Peake. Architecture-based fault tolerance support for grid applications. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems–ISARCS*, pages 177–182. ACM, 2011.

[ZL12] Albert Y Zomaya and Young Choon Lee. *Energy Efficient Distributed Computing Systems*, volume 88. John Wiley & Sons, 2012.

[ZMSM10] Wenbing Zhao, PM Melliar-Smith, and Louise E Moser. Fault tolerance middleware for cloud computing. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 67–74. IEEE, 2010.

[ZQQ11] Xiaomin Zhu, Xiao Qin, and Meikang Qiu. Qos-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters. *Computers, IEEE Transactions on*, 60(6):800–812, 2011.

[ZSB+12] Sergey Zhuravlev, Juan Carlos Saez, Sergey Blagodurov, Alexandra Fedorova, and Manuel Prieto. Survey of scheduling techniques for addressing shared resources in multicore processors. *ACM Computing Surveys (CSUR)*, 45:4, 2012.

[ZT12] Qian Zhu and Teresa Tung. A performance interference model for managing consolidated workloads in qos-aware clouds. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 170–179. IEEE, 2012.

[ZW11] Tianhai Zhao and Yunlan Wang. Virtual high performance computing environments for science computing on-demand. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 136–140. IEEE, 2011.

[ZZA10] Qian Zhu, Jiedan Zhu, and Gagan Agrawal. Power-aware consolidation of scientific workflows in virtualized environments. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. IEEE Computer Society, 2010.