Energy-Efficient Management of Resources in Container-based Clouds

Sareh Fotuhi Piraghaj

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

March 2016

Department of Computing and Information Systems
The University of Melbourne, Australia

Energy-Efficient Management of Resources in Container-based Clouds

Sareh Fotuhi Piraghaj
Principal Supervisor: Prof. Rajkumar Buyya
Co-Supervisor: Dr. Rodrigo N.Calheiros

Abstract

CLOUD enables access to a shared pool of virtual resources through Internet and its adoption rate is increasing because of its high availability, scalability and cost effectiveness. However, cloud data centers are one of the fastest-growing energy consumers and half of their energy consumption is wasted mostly because of inefficient allocation of the servers resources. Therefore, this thesis focuses on software level energy management techniques that are applicable to containerized cloud environments. Containerized clouds are studied as containers are increasingly gaining popularity. And containers are going to be major deployment model in cloud environments.

The main objective of this thesis is to propose an architecture and algorithms to minimize the data center energy consumption while maintaining the required Quality of Service (QoS). The objective is addressed through improvements in the resource utilization both on server and virtual machine level. We investigated the two possibilities of minimizing energy consumption in a containerized cloud environment, namely the VM sizing and container consolidation. The key contributions of this thesis are as follows:

1. A taxonomy and survey of energy-efficient resource management techniques in PaaS and CaaS environments.
2. A novel architecture for virtual machine customization and task mapping in a containerized cloud environment.
3. An efficient VM sizing technique for hosting containers and investigation of the impact of workload characterization on the efficiency of the determined VM sizes.
4. A design and implementation of a simulation toolkit that enables modeling for containerized cloud environments.
5. A framework for dynamic consolidation of containers and a novel correlation-aware container consolidation algorithm.
6. A detailed comparison of energy efficiency of container consolidation algorithms with traditional virtual machine consolidation for containerized cloud environments.

Declaration

This is to certify that

1. the thesis comprises only my original work towards the PhD,
2. due acknowledgement has been made in the text to all other material used,
3. the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.

Sareh Fotuhi Piraghaj, 24 March 2016

Preface

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2- 5 and are based on the following publications:

- **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros, and Rajkumar Buyya, "A Survey and Taxonomy of Energy Efficient Resource Management Techniques in Platform as a Service Cloud," *Handbook of Research on End-to-End Cloud Computing Architecture Design Book, J.Chen, Y.Zhang, and R.Gottschalk (eds), IGI Global, Pages. 410 - 454. Web. 16 Oct. 2016. doi: 10.4018/978-1-5225-0759-8.ch017, Hershey, PA, USA, 2017.*

- **Sareh Fotuhi Piraghaj**, Rodrigo N.Calheiros, Jeffery Chan, Amir Vahid Dastjerdi , and Rajkumar Buyya "A Virtual Machine Customization and Task Mapping Architecture for Energy Efficient Allocation of Cloud Data Center Resources," *The Computer Journal, vol. 59, no. 2, Pages. 208 - 224, ISSN 0010-4620, Oxford University Press, UK, November, 2015.*

- **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "Efficient Virtual Machine Sizing For Hosting Containers as a Service," *Proceeding of the 2015 IEEE World Congress on Services (SERVICES2015)*, Pages. 31 - 38, New York, United States.

- **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "An Environment for Modeling and Simulation of Containers in Cloud Data

Centers", *Software: Practice and Experience (SPE)*, John Wiley & Sons, Ltd, USA, 2016 . [Online]. Available: http://dx.doi.org/10.1002/spe.2422.

- **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers ," *Proceedings of the 11th IEEE International Conference on Green Computing and Communications (GreenCom 2015)*, Pages: 368 - 375, Sydney, Australia, 2015.

Acknowledgements

Reflect upon your present blessings, of which every man has plenty; not on your past misfortunes, of which all men have some. – Charles Dickens

PhD is a rewarding journey, which would not be possible without the support of many people. As my journey is near to its end, I would like to take this opportunity to thank these amazing people who inspired me during the ups and downs of this pleasant experience.

First and foremost, I would like to express my sincere gratitude to my principal supervisor, Professor Rajkumar Buyya for giving me the opportunity to pursue my studies in his eminent group. His continuing guidance, support, and encouragement helped me in all aspects of my research and writing of this dissertation. Secondly, I would like to acknowledge my co-supervisor, Doctor Rodrigo N.Calheiros, on his precious support and wise advice that made the contributions of this thesis more significant.

I would like to express my appreciation to my collaborator Dr. Jeffery Chan for his valuable and insightful comments on the third chapter of this thesis. I also thank Dr. Amir Vahid Dastjerdi, for his generous guidance on developing research skills, collaborating on research, providing constructive comments and proofreading the dissertation. I thank Professor Christopher Andrew Leckie for serving as the chair of the PhD committee and offering his constructive feedback on my research work.

I would like to thank all past and current members of the CLOUDS Laboratory, at the University of Melbourne: Atefeh Khosravi, Adel Nadjaran Toosi, Yaser Mansouri, Maria Rodriguez, Chenhao Qu, Yali Zhao, Jungmin Jay Son, Bowen Zhou, Farzad Khodadadi, Hasanul Ferdaus, Safiollah Heidari, Liu Xunyun, Caesar Wu, Minxian Xu, Sara Kardani Moghaddam, Muhammad H.Hilman, Redowan Mahmud, Anton Beloglazov, Nikolay Grozev, Deepak Poola, Mohsen Amini Salehi, Saurabh Garg, and Mohammed Alrokayan for their friendship and support.

I acknowledge the University of Melbourne and the Australian Research Council (ARC) grants (awarded to my principal supervisor) for providing scholarships and facilities to pursue my research. I am also thankful for Amazon Web Services (AWS) research grant that provided me a real cloud environment for running experiments and validating my research assumptions. I also thank the CIS Department administrative staff members Rhonda Smithies, Madalain Dolic, and Julie Ireland and Professor Justin Zobel for their support and guidance.

I express my profound gratitude to my parents, who have always been supporting me in every stage of life including my undergraduate and postgraduate studies. I also thank my sisters, and brother for their love and encouragements in time of trouble and doubt. Your prayer for me was what sustained me thus far.

I thank my brother- and sisters-in-law for their precious understanding and encouragement. I specially thank my parents-in-law for their support and thoughtfulness over these years.

Lastly and most importantly, I would like to dedicate this thesis to my beloved husband, Maysam, who has been making my life so incredible and prosperous each and every day. These few words can not express my deepest appreciation for his selfless patience, unconditional love, and endless support during these past years.

Sareh Fotuhi Piraghaj
Melbourne, Australia
March 24, 2016

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CLOUD computing is a realization of utility-oriented delivery of computing services on a pay-as-you-go basis [22]. There are a variety of definitions of Cloud Computing and the specific characteristics it offers to a user. The National Institute of Standards and Technology (NIST) [112] defines Cloud Computing as "... *a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction*". As stated by Armbrust et al. [6], cloud computing has the potential to transform a large part of the IT industry while making software even more attractive as a service.

Traditional cloud services are broadly divided into three service models, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In the IaaS service model, a cloud customer has the ability to provision virtualized resources using both web portals and APIs. Gartner [1] defines IaaS as "... *a standardized, highly automated offering, where compute resources, complemented by storage and networking capabilities are owned and hosted by a service provider and offered to customers on-demand*". The PaaS service model has a higher level of abstraction when compared to IaaS. This service model enables developers to build applications and services over the Internet by providing a platform and an environment that is accessible by the web browser. Server-side scripting environment, database management system, and server software are some of the features that can be included in the PaaS cloud service model [2]. The Software as a Service (SaaS) cloud model enables customers to access applications over the Internet [3].

[1]http://www.gartner.com/it-glossary/infrastructure-as-a-service-iaas/
[2]http://www.interoute.com/what-paas
[3]http://www.interoute.com/what-saas

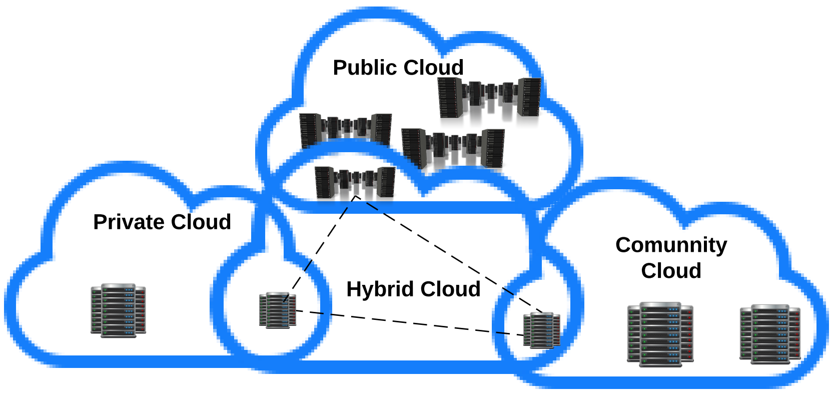


Figure 1.1: The four cloud deployment models: private, public, community, and hybrid cloud.

Google docs, Facebook, and Twitter are some examples of this cloud service model.

In addition to these service models, Clouds are categorized into four deployment models on the basis of their availability to the general public (Figure 1.1):

- **Public:** In this model, the Cloud is available to the general public.

- **Private:** The private Cloud is accessible by a business or organization while it is not available to the general public.

- **Community:** This category of cloud provides services to a limited number of individuals or organizations that have shared concerns (e.g., mission, security requirements, and compliance considerations). These organizations are commonly managed, secured, and governed by either a third party managed service provider or all of the participating organizations.

- **Hybrid:** This model is an integration of two or more of the aforementioned cloud deployment models. Here, customers benefit from the multiple deployment models, what consequently eliminates the boundaries and limitations of each cloud model while increasing the capacity through aggregation [4].

[4]https://www.ibm.com/developerworks/community/blogs/722f6200-f4ca-4eb3-9d64-8d2b58b2d4e8/entry/4_Types_of_Cloud_Computing_Deployment_Model_You_Need_to_Know1?lang=en

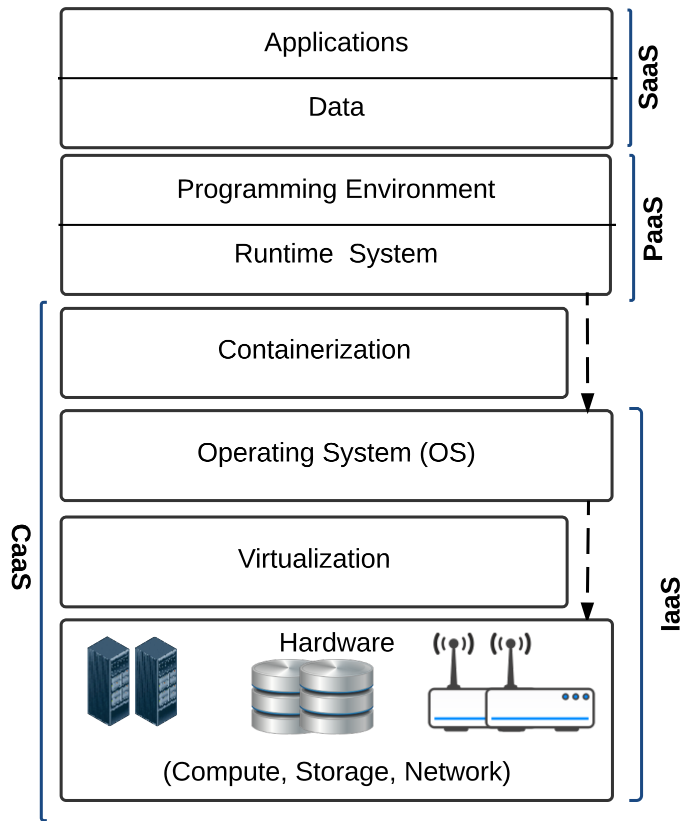


Figure 1.2: The Container as a Service cloud service model links the PaaS and IaaS layers.

1.1 Container as a Service Cloud Deployment Model

PaaS provides a platform for application development that allows users not to worry about the underlying infrastructure and technologies. These platforms are widely used by companies for various purposes such as hosting mobile systems [105]. Google App Engine (GAE) and Amazon Web Services (AWS) Elastic Beanstalk are examples of PaaS services. Gartner named 2015 as "*the year of PaaS.* " [5] and defined Platform-as-a-Service (PaaS) as "*a broad collection of application infrastructure (middleware) services (including application platform, integration, business process management and database services)*"[6].

Despite the advantages of the PaaS service model, there are still a number of drawbacks that has limited the broader adoption of this service model. PaaS provides a platform that is optimized for a specific use case, while IaaS gives costumers flexibility by of-

[5] http://insights.wired.com/profiles/blogs/why-2015-is-the-year-of-paas#axzz3s5dYcebL
[6] http://www.gartner.com/it-glossary/platform-as-a-service-paas

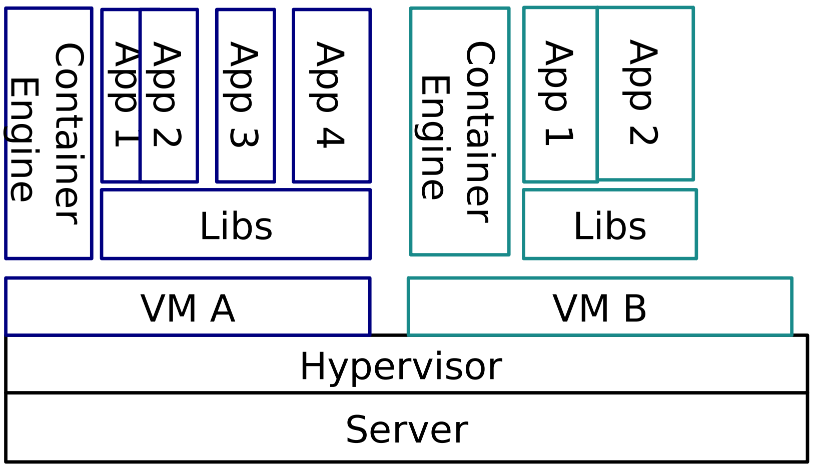


Figure 1.3: A simple CaaS deployment model on IaaS.

fering configurable virtual environments (virtual machines). By utilizing the PaaS model, customers are able to focus on the code only, without being concerned about maintenance costs, runtime environments, and operating systems. However, applications developed in a PaaS environment are restricted by the platform specifications. For example, in order to be able to run Java applications on GAE, developers must first make sure that their utilized third party libraries are compatible with GAE. This is because GAE does not support all the Java Runtime Environment (JRE). In this respect, CaaS (Container as a Service) is introduced to solve these issues resulted from the dichotomy between the IaaS and PaaS models [143]. Containers, as the building blocks of the CaaS cloud model, offer isolated virtual environments without requiring intermediate monitoring media such as hypervisors. Containers increase the efficiency of cloud resource utilization as they are denser compared to virtual machines. In addition, as containers share the host Operating System kernel, their communication is performed via system standard calls, which is much faster than hypervisor-based communication of virtual machines.

Amazon EC2 Container Service (ECS) and Google Container Engine are two examples of CaaS cloud environments that lie between IaaS and PaaS. While IaaS provides virtualized compute resources and PaaS provides application specific runtime services, CaaS is the missing layer that links these two layers together (as depicted in Figure 1.2). CaaS services are usually provided on top of IaaS' virtual machines, as illustrated in Figure 1.3. CaaS providers, such as Google and AWS, argue that while containers offer appropriate environment for semi-trusted workloads, virtual machines provide another layer of security for untrusted workloads. While Google runs containers on bare-metal in its private infrastructure, this option is not available in public cloud environment. In addition to this, virtual machines enable the system load optimization where containers are not using the whole physical server's capacity. Apart from this, infrastructure owner will not be able to run mix workloads while running containers on bare-metal. However, this is not the case for VMs as VMs with different operating systems can run on one physical server. Moreover, virtual machines are more advanced in terms of disaster recovery when compared with physical servers. VMs also enable multi tenancy when the workloads can not share the same kernel. Apart from this, VM provisioning is a lot easier through API when compared to physical server provisioning, therefore VMs are also beneficial for automation purpose[7]. On the other hand in a CaaS environment, Containers decouple applications from their running environment and consequently eliminate platform-dependency [7], which is one of the drawbacks of PaaS. Containerization brings portability so developers are able to build applications and run them anywhere on any kind of platform. The CaaS cloud model is considered a gateway to active application management.

1.2 Energy Consumption Challenges in Containerized Clouds

The numerous advantages of cloud computing environments, including cost effectiveness, on-demand scalability, and ease of management, encourage service providers to adopt them and offer solutions via cloud models. This in turn encourages platform providers to increase the underlying capacity of their data centers to accommodate the increasing demand of new customers. One of the main drawbacks of the growth in capacity of cloud data centers is the need for more energy to power these large-scale infrastructures. Such a drastic growth in energy consumption of cloud data centers is a major concern of cloud providers.

An average data center consumes as much energy as 25,000 households, as reported by Kaplan et al. [90]. This energy consumption results in increased Total Cost of Own-

[7]https://blog.docker.com/2016/04/physical-virtual-container-deployment/

ership (TCO) and consequently decreases the Return of Investment (ROI) of the cloud infrastructure. Apart from low ROI, energy consumption has a significant impact on carbon dioxide (CO2) emissions, which are estimated to be 2% of global emissions [21].

Energy wastage in data centers are driven by various reasons such as inefficiency in data center cooling systems [71], network equipments [80], and server utilization [70]. However, servers are still the main power consumers in a data center [70]. Both the amount of work and the efficiency with which the work is performed affect the power consumption of servers [103]. Therefore, for improving the power efficiency of data centers, the energy consumption of servers should be made more proportional to the workload. Power proportionality is defined as the proportion of the amount of power consumed comparing to the actual workload and it can be achieved by either decreasing servers' idle power utilization at hardware level [11] or efficient provisioning of servers through power-aware resource management policies at software level.

Although there is a large body of research on energy efficient resource management of IaaS, not enough attention has been given to PaaS environments with containers. Hence, this thesis focuses on software-level energy management techniques that are applicable to containerized cloud environments. The main objective is improving data center energy consumption while maintaining the required Quality of Service (QoS) through decreasing Service Level Agreement (SLA) violations. This thesis contributes to the literature by considering containerized cloud environments while addressing their new challenges. One of the aspects that distinguishes this thesis from the related work is that this thesis tackles the problem of data center energy consumption through the study of real cloud backend data. It also explores the potential benefits, for containerized cloud environments, from a comprehensive cloud workload study and how it can decrease the amount of energy consumption in the data center.

1.3 Research Problems and Objectives

This thesis tackles research challenges in relation to energy-efficient resource management techniques applicable for containerized cloud environments in which containers are running on VMs. In summary, the following research problems are explored:

- **How to map tasks/containers to virtual machines considering available cloud workload data?**

  The determination of virtual machine configuration is an important factor that affects the amount of resources required for task/container placement and the total energy consumption of a data center. Considering a single VM size for each container individually increases the chance of system fragmentation. In this respect, classification of containers into different groups has the potential to decrease the granularity of the placement problem. Apart from this, the analysis of usage patterns of each class of containers can help in the determination of the optimal number of containers that can be hosted in one VM.

- **How workload characterization methodologies/techniques affect the number of identified virtual machine sizes and energy consumption in a PaaS/ CaaS environment?**

  As the output of the workload characterization step is utilized in the determination of virtual machine sizes, it is essential that the methodology used to characterize the workload is well understood. This is required to avoid over classification of the workload, which may increase both resource wastage and fragmentation. Hence, making sure that the workload analysis results in an efficient resource allocation is as important as the characterization process itself.

- **How the algorithms applied in various stages of the consolidation process in a CaaS/PaaS environment affects the total energy consumption and SLA violations?** Similarly to any consolidation problem, the container consolidation problem should also be considered as a multi-stage problem as follows.

  1. **When to trigger the migration?**

     Container migration is triggered when a host is found to be overloaded or underloaded. When they are overloaded, the migration happens to avoid further SLA violations and performance degradation. However, when they are underloaded, the migration objective is improving the resource utilization by migrating containers away from underloaded hosts so that they can be switched off or put in a lower power state.

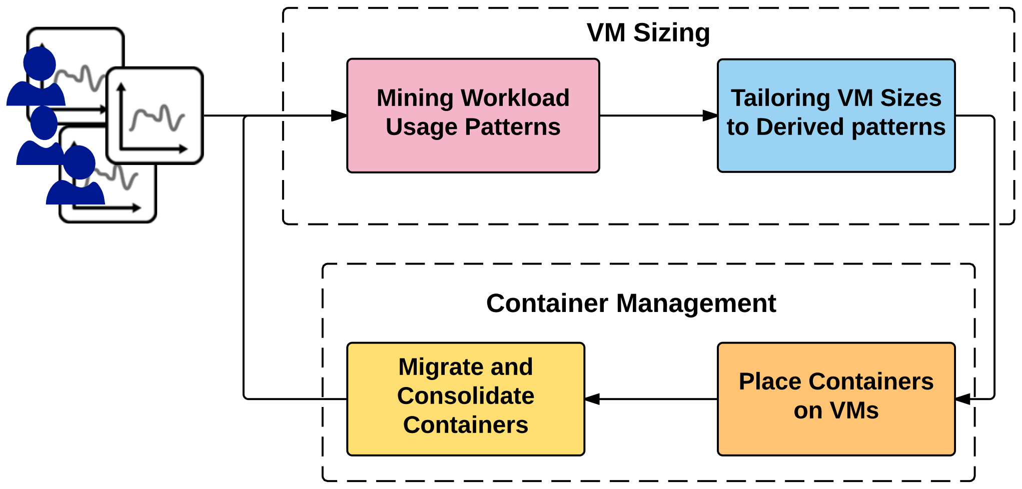

VM Sizing
Mining Workload Usage Patterns
Tailoring VM Sizes to Derived patterns
Container Management
Migrate and Consolidate Containers
Place Containers on VMs

Figure 1.4: Outline of the thesis objective.

2. **Which containers to migrate?**

   When a host is found to be overloaded, it is the time to choose a set of containers to move from the overloaded hosts to resolve the situation. As host overload increases incurred SLA violations, the container selection criteria may affect the energy efficiency of the consolidation approach and the performance of applications running inside containers.

3. **Where to migrate?**

   When the set of containers are selected for migration, it is important to find the best placement for them. Selected migration destinations affect the efficiency of the consolidation process in terms of energy efficiency and SLA violations. For instance, selecting the most utilized host as the migration destination might increase SLA violations, since the probability that the most utilized host will experience overload in the near future is higher than for the least utilized host.

- **Is container consolidation a better approach than VM consolidation?**

  In CaaS environments, there is the option of consolidation at VM and container levels. This option raises the question of which one is more beneficial in terms of total energy consumption of the data center along with incurred SLA violations.

Considering the aforementioned research problems, the following **objectives** are identified (Figure 1.4):

- Explore, analyze, and systematize research in the area of PaaS/CaaS energy-efficient resource management techniques to obtain an understanding of the studied approaches along with the shortcomings of existing applied algorithms.
- Conduct a brief analysis of the only publicly available cloud backend traces, released by Google in 2011, to obtain insights about the challenges in a real cloud environment.
- Propose a task mapping and VM size customization architecture that is inspired by the Google workload analysis and characterization results. This architecture benefits from both virtual machine and containerization technology.
- Explore the effect of workload analysis on the efficiency of customized VM sizes considering data center' total energy consumption and container/task rejection rates.
- Compare the efficiency of customized VM types with the currently offered VM configurations considering data center's total energy consumption.
- Extend the current available cloud simulator CloudSim to model the CaaS cloud environment, which enables the ability to conduct experiments and compare container/VM consolidation approaches in a controllable and reproducible way.
- Investigate the effect of the algorithms applied in different stages of the consolidation process on data center's total energy consumption and incurred SLA violations.
- Propose a correlation-aware container consolidation algorithm and compare its efficiency with the efficiency of available approaches.
- Compare the efficiency of VM consolidation approaches with container consolidation algorithms in terms of data center's energy consumption and incurred SLA violations.

1.4 Research Methodology

The virtual machine sizing and task mapping approaches presented in this thesis are evaluated using real cloud backend traces. For this purpose, we analyzed and characterized the publicly available Google cloud backend traces. The traces were released in two versions. The first Google log provides normalized resource usage of a set of tasks over a 7-hour period. The second version of the Google traces, released in 2012, contains more details in a longer time frame. Therefore, the data set used in this thesis is derived from the second version of the Google cloud trace log [137] collected during a period of 29 days. The log consists of data tables describing machines, jobs, and tasks.

In addition, to evaluate the proposed container and VM consolidation algorithms, we utilized simulated application workloads obtained from real-world workload traces. The CPU utilization of containers are simulated based on the data provided by Beloglazov et al. [15]. The workload data was obtained from monitored CPU utilization by the CoMon project of the PlanetLab [127] infrastructure.

In order to enable evaluation of our proposed algorithms, we utilized simulation. For this purpose, we designed and implemented a simulator that enables modeling of containerized cloud environments. Simulation was selected as the evaluation methodology because it provides a repeatable and controllable environment to evaluate our proposed algorithms. The proposed simulator for containerized environment enables researchers to plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance.

1.5 Research Contributions

This thesis' contributions can be broadly classified into 5 major categories, namely analysis and classification of prior research work, presentation of an efficient virtual machine customization and task mapping architecture for a containerized cloud environment, proposition of virtual sizing techniques for CaaS cloud service model, implementation of the ContainerCloudSim simulator for containerized cloud environments, and investigation of energy efficiency of container consolidation algorithms. The **key contributions** of this thesis are:

1. A taxonomy and survey of energy-efficient resource management techniques in PaaS and CaaS environments.

2. Virtual machine customization and task mapping architecture.

    - A detailed analysis of Google trace workload, which is the only publicly available cloud backend data.
    - An end-to-end architecture for efficient allocation of tasks on data centers that decreases data center's total energy consumption .
    - Identification of virtual machine configurations sizes (types) in terms of CPU, memory, and disk capacity through extraction of resource utilization patterns of tasks.
    - Determination of the maximum number of tasks (task capacity) that can be accommodated in each virtual machine type. Various estimates such as average resource usage of tasks in each cluster are considered for this purpose.

3. Efficient VM sizing technique for hosting Containers as a Service:

    - An approach to determine the most efficient VM sizes considering similarities in usage patterns of tasks.
    - Investigation of the impact of the feature set selection on the number of resulting clusters and resource allocation efficiency.
    - A comparison of our VM sizing technique with fixed VM size baseline scenarios.

4. Design and implementation of the "ContainerCloudSim" simulator, which enables modeling of containerized cloud environments.

    - A simulation architecture for containerized clouds and its implementation. The simulator provides a repeatable and controllable environment that supports modeling and simulation of containerized cloud computing environments.

- A number of use cases to demonstrate how researchers can plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance.
- Evaluation of the scalability of our system as it supports simulation of large number of containers.
- Modeling of the startup delay of containers considering real experiments while incorporating it in the simulation.

5. Novel algorithm for energy efficient container consolidation.

    - A framework for dynamic container consolidation.
    - A novel correlation-aware container consolidation algorithm.
    - A comparison of the heuristics used in different stages of VM and container consolidation considering the total energy consumption of the data center and SLA violations.

1.6 Thesis Organization

The core chapters of this thesis are derived from several journal and conference papers during the PhD candidature. The thesis structure is depicted in Figure 1.5. Chapter 3 is focused on the virtual machine sizing techniques for containerized cloud environments. Chapter 4 and Chapter 5 are focused on container consolidation. The remainder of the thesis is organized as follows:

- Chapter 2 presents a taxonomy and survey of energy management techniques applied in cloud computing environments with focus on both Platform as a Service (PaaS) and Container as a Service (CaaS) cloud deployment models. In addition, this chapter also contains the scope of the thesis along with its positioning in the literature. This chapter is derived from:

    – **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros, and Rajkumar Buyya, "A Survey and Taxonomy of Energy Efficient Resource Management Techniques in Platform as a Service Cloud," *Handbook of Research*

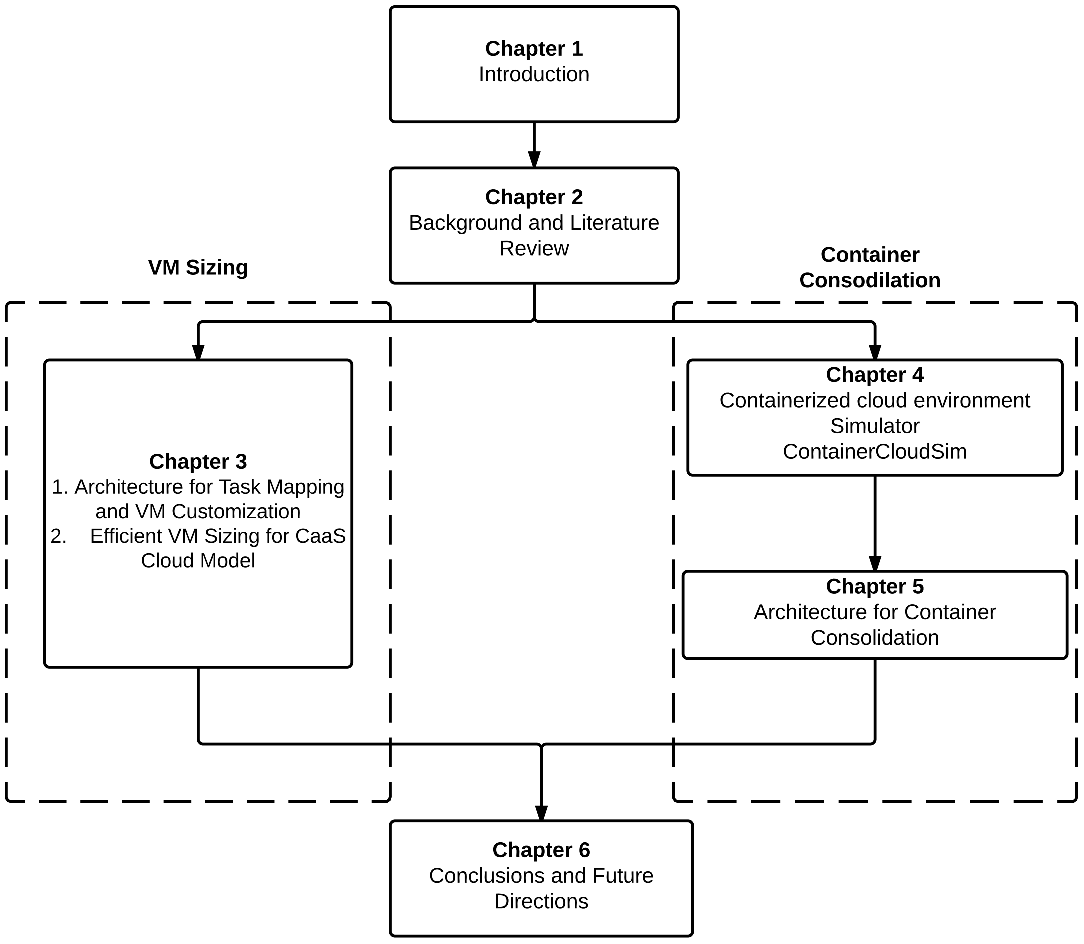

Chapter 1
Introduction
Chapter 2
Background and Literature Review
VM Sizing
Container Consodilation
Chapter 3
1. Architecture for Task Mapping and VM Customization
2. Efficient VM Sizing for CaaS Cloud Model
Chapter 4
Containerized cloud environment Simulator ContainerCloudSim
Chapter 5
Architecture for Container Consolidation
Chapter 6
Conclusions and Future Directions

Figure 1.5: Thesis organization.

on End-to-End Cloud Computing Architecture Design Book, J.Chen, Y.Zhang, and R.Gottschalk (eds), IGI Global, Pages. 410 - 454. Web. 16 Oct. 2016. doi: `10.4018/978-1-5225-0759-8.ch017`*, Hershey, PA, USA, 2017.*

- Chapter 3 presents an architecture for efficient allocation of cloud resources in the Platform as a Service cloud service model. Workload characterization is leveraged to determine the most efficient virtual machine configurations for each group of tasks. In addition to VM configuration, various algorithms are proposed for consolidation of tasks/containers on VMs and the results are compared in terms of data center energy consumption and task rejection rate. Real cloud backend data released by Google is characterized and used as system input for validation purposes. In addition, the chapter also carries out an investigation of the efficiency of the obtained VM configurations when compared to some of the available VM sizes in common cloud providers such as Amazon EC2. The effect of the workload characterization is then studied considering two different feature sets. This chapter is derived from:
  - **Sareh Fotuhi Piraghaj**, Rodrigo N.Calheiros, Jeffery Chan, Amir Vahid Dastjerdi , and Rajkumar Buyya "A Virtual Machine Customization and Task Mapping Architecture for Energy Efficient Allocation of Cloud Data Center Resources," *The Computer Journal, vol. 59, no. 2, Pages. 208 - 224, ISSN 0010-4620, Oxford University Press, UK, November, 2015.*
  - **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "Efficient Virtual Machine Sizing For Hosting Containers as a Service," *Proceeding of the 2015 IEEE World Congress on Services (SERVICES2015)*, Pages. 31 - 38, New York, United States.
- Chapter 4 describes the ContainerCloudSim simulator. This software enables researchers to validate their proposed container consolidation algorithms and also compare them with VM consolidation algorithms in a simulated environment. This chapter is derived from:
  - **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros, and Ra-

jkumar Buyya, "An Environment for Modeling and Simulation of Containers in Cloud Data Centers", *Software: Practice and Experience (SPE)*, 2016. [Online]. Available: http://dx.doi.org/10.1002/spe.2422.

- Chapter 5 presents an architecture for consolidation of containers on virtual machines in CaaS environments. It presents a correlation-aware algorithm for packing containers and compares the efficiency of this approach with commonly used packing algorithms. It also contains a brief comparison of the algorithms utilized in different stages of the consolidation process for containers. Furthermore, the efficiency of container consolidation is compared with virtual machine consolidation in terms of energy consumption and SLA violations.This chapter is derived from:
  - **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers ," *Proceedings of the 11th IEEE International Conference on Green Computing and Communications (GreenCom 2015)*, Pages: 368 - 375, Sydney, Australia, 2015.
- Chapter 6 concludes the thesis with a summary of the main findings, an outline of possible future research directions, and final remarks.

Chapter 2

Literature Survey and Related Work

The numerous advantages of cloud computing environments, including scalability, high availability, and cost effectiveness have encouraged service providers to adopt the available cloud models to offer solutions. This rise in cloud adoption, in return encourages platform providers to increase the underlying capacity of their data centers so that they can accommodate the increasing demand of new customers. Increasing the capacity and building large-scale data centers has caused a drastic growth in energy consumption of cloud environments. The energy consumption not only affects the Total Cost of Ownership but also increases the environmental footprint of data centers as CO2 emissions increases. Hence, energy and power efficiency of the data centers has become an important research area in distributed systems. In order to identify the challenges in this domain, this chapter surveys and classifies the energy efficient resource management techniques specifically focused on the PaaS and CaaS cloud service models. Finally, the chapter concludes with a brief discussion about the scope of the current thesis along with its positioning within the research area.

2.1 Introduction

DATA centers, as the backbone of the modern economy, are one of the fastest-growing power consumers [37]. U.S. data centers consumed of 75 billion kWh of electricity annually which was equivalent to the output of around 26 medium-sized coal-fired power plants. This energy usage is estimated to reach 140 billion kilowatt-hours annually, in the next four years [37]. Despite this huge amount of energy that is required to power on these data centers, half of this energy is wasted mostly due to the inefficient allocation of servers resources.

This chapter is derived from: **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros, and Rajkumar Buyya, " A Survey and Taxonomy of Energy Efficient Resource Management Techniques in Platform as a Service Cloud," *Handbook of Research on End-to-End Cloud Computing Architecture Design Book, J.Chen, Y.Zhang, and R.Gottschalk (eds), IGI Global, Pages. 410 - 454. Web. 16 Oct. 2016. doi: 10.4018/978-1-5225-0759-8.ch017, Hershey, PA, USA, 2017.*

The energy wastage not only increases the electricity bills, it is also considered a threat for the environment, as it contributes to the global warming phenomena and increases the CO2 emissions. In this respect, there has been an increasing effort to minimize the power consumption of cloud data centers through various energy efficient resource management techniques. In this chapter we survey the research applied resource management techniques in this area while focusing more on the PaaS and containerized cloud models. These models are selected as they are the main focus of the current thesis.

2.2 PaaS Power-aware Resource Management

There is a large body of literature investigating energy management techniques for PaaS cloud service model that provides a platform for cloud customers to develop, run, and manage their applications without worrying about the underlying infrastructure and the required software. Both kinds of virtualization namely, OS level and System level virtualization, are considered and the newly introduced CaaS model can be viewed as a form of OS level virtualization service. Since CaaS cloud model has been newly introduced, we grouped all the research with the focus on containerized (OS-level virtualized) cloud environments under the PaaS category.

The work in this area, as demonstrated in Figure 2.1, is grouped in two major categories namely Bare Metal, non-virtualized, and Virtualized. The *Bare Metal* group contains the techniques in which the applications/tasks are mapped to the servers without considering virtualization technology, whereas the work investigating energy efficient techniques in a virtualized environment are all included in *Virtualized* group.

2.2.1 Bare Metal Environments

Servers are one of the most power-hungry elements in data centers, with CPU and memory as their main power consumers. The average power consumption of CPU and memory is reported to be 33% [111] and 23% [35] of the server's total power consumption respectively. Therefore, any improvement on processor and memory-level power consumption would definitely reduce the total power consumption of the server, which also improves the energy efficiency of data center.

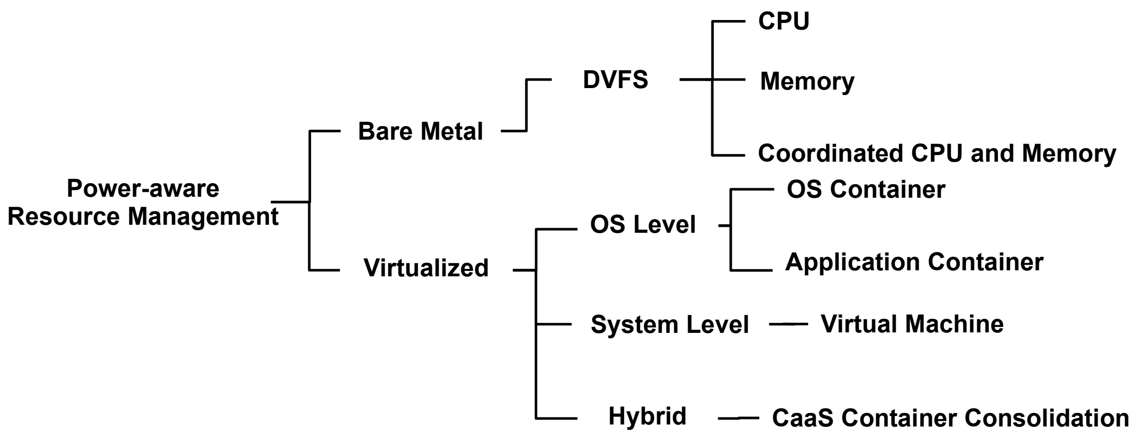


Figure 2.1: Power-aware PaaS resource management research breakdown

Dynamic Voltage and Frequency Scaling (DVFS) is an effective system level technique utilized both for memory and CPU in *Bare Metal* environments and it is demonstrated to improve the power consumption of these two elements [28, 35, 39, 160] considerably. DVFS enables dynamic power management through varying the supply voltage or the operating frequencies of the processor and/or memory. Research in this area are summarized in Tables 2.3 - 2.5.

Dynamic Voltage and Frequency Scaling of CPU

The technologies present in the market are AMD Turbo Core[1], Intel Turbo Boost [28], and Intel Enhanced Speed Stepping Technology[2], which dynamically adjust the CPU frequency and voltage according to the workload. Kim et al. [99], harnessed the DVFS capability of CPU in the proposed scheduling algorithm. DVS scheduling scheme considers the deadline of the Bag-of-Tasks applications as a constraint and the CPU frequency is adjusted so that the sub-tasks are finished by the deadline. An application made of a group of independent and identical tasks is an example of Bag-of-Task applications. DVS scheduling algorithms are provided for both time-shared and space-shared resource sharing policies. Proposed algorithm is validated through simulation and is shown to be more energy efficient when compared to the static voltage schemes.

Pietri et al. [133] also proposed an energy efficient scheduling algorithm utilizing the

[1]http://www.amd.com/en-us/innovations/software-technologies/turbo-core
[2]http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm#overview

DVFS capability of CPU. The frequency of the CPU is adjusted with the objective of reducing the total energy consumption for the execution of tasks while meeting a user-specified deadline. Decreasing the overall energy consumption is considered as the objective of the algorithms, since DVFS is not always energy efficient, as scaling the CPU frequency may increase the execution time. Hence, it escalates the processors idle time. Based on the aforementioned objective, it is demonstrated that the lowest possible frequency is not always the most energy-efficient option. Therefore, the proposed approach only scales the frequency if the overall energy consumption can be minimized.

Dynamic Voltage and Frequency Scaling of Memory

In addition to CPU, memory of servers also consumes a considerable amount of energy that is not proportional to the load [35]. For memory-intensive workloads, system's memory speed is well tuned and optimized according to the peak computing power. However, there is still a place for improvement for other kinds of workloads that are less sensitive to the memory speed. For these kinds of workload, running at lower memory speed would result in less performance degradation and reduce the power consumption via running memory at a lower frequency.

David et al. [35] presented an approach utilizing the memory DVFS capability to tune the system's memory frequency based on the workload and consequently minimize the energy consumption. Additionally, a detailed power model is presented which quantifies dependency portions of memory power to the frequency and further proves the possibility of considerable power deduction through memory DVFS. Also, a control algorithm is proposed to tune frequency/voltage of memory considering its bandwidth utilization with the objective of minimizing performance degradation. The approach is evaluated through implementation on real hardware while SPEC CPU2006 is used to generate the workload. This work can further be extended for different types of workloads considering DVFS application for both CPU and memory components.

Deng et al. [39] introduces active lower-power modes (MemScale) for main memory to make it more energy proportional. In this respect, DVFS and dynamic frequency scaling (DFS) are applied on the memory controller and its channels and DRAM devices, respectively. MemScale is implemented as an operating system policy and, like David et al. [35], it identifies the DVFS/DFS mode for the memory subsystem according to the bandwidth utilization of memory. The objective of the research is also similar to the work by David et al. [35], which improves the energy consumption of the memory subsystem. This is important because it can reach up to 40% of the system's energy utilization [82]. MemScale is evaluated through simulation considering a large set of workloads with less than 10% performance degradation while in [35] only one workload is studied.

Coordinated CPU and Memory DVFS

Deng et al.[38] introduced CoScale, which jointly applies DVFS on memory and CPU subsystems with the objective of minimizing the systems total power consumption. CoScale is the first work in this area that coordinates DVFS on CPU and memory considering performance constraints. The frequency of each core and the memory bus is selected in a way that energy saving of the whole system is maximized. Therefore, the selected frequencies are not always the lowest ones.

As observed by Dhimsan et al. [40], lowering the frequency sometimes results in more energy consumption. So CoScale always balances the system and component power utilization. It efficiently searches the space of available frequency settings of CPU and memory and sets the components voltage according to the selected frequencies. In this respect, the algorithm should consider $m * n * c$ possibilities in which m and c are the number of available frequency setting for memory and CPU respectively and n is the number of CPU cores. In order to accelerate the search process, a gradient-descent heuristic is proposed that iteratively estimates the frequencies of the components through the presented online models. Memory-intensive (MEM), compute-intensive (ILP), compute-memory balanced (MID), and a combination of workloads are applied as the input of the system. The results of CoScale is further compared with four different algorithms, namely MemScale [39], CPU DVFS, a fully uncoordinated, and a semi-coordinated algorithm. In the fully uncoordinated algorithm, both memory and CPU frequency are decided by their managers independently. In semi-coordinated policy, the CPU manager is aware of the degradation caused by the memory manager decision in the previous cycle through accessing overall performance slack. CoScale satisfies the performance target while being robust-across the search space parameter.

Table 2.1: Hardware Virtualization Taxonomy.

Virtualization	Component	Communication with Hardware	Available Technologies
Operating System (OS)	OS Container (Lightweight VM)	System Standard Calls	LXC, OpenVZ, Linux VServer, FreeBSD Jails, Solaris zones
	Application Container		Docker, Rocket
System	Virtual Machine (VM)	Hypervisor	KVM, VMWare

2.2.2 Virtualized Environments

Virtualization technology is one of the key features in cloud data centers that can improve the efficiency of hardware utilization through resource sharing, migration, and consolidation of workloads. The technology was introduced in the 1960's [68, 69] and exists in many levels. Of interested in this chapter, are virtualization at operating system level and at system level (Table 2.1).

In system-level virtualization, there exists the emulated hardware referred as 'virtual machines' (VMs) that have their own operating system (OS) running on top of the host's hypervisor with independent kernels. However, on the operating system level, there exists the so called containers that share the same kernel with the host and are defined as lightweight virtual environments that provide a layer of isolation between workloads without the overhead of the hypervisor-based virtualization.

Considering these two virtualization types, techniques investigating power-aware resource management are divided into three main categories namely Lightweight Container, Virtual Machine, and Hybrid. These groups are formed according to the environment in which applications execute. Therefore, the *Lightweight container* category contains techniques that assume that tasks/applications execute inside containers. In the *Virtual Machine* group, applications execute inside virtual machines. Finally in the *Hybrid* category, applications execute inside the containers while containers are mapped on virtual machines instead of servers.

Next, we discuss these three groups with more details and explore techniques that are

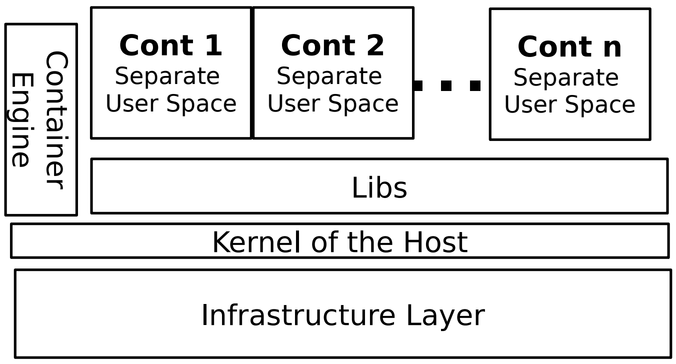


Figure 2.2: Containerized Virtual Environment

applied to minimize the data center energy consumption considering the characteristics of each virtualized environment. The research in this area are summarized in Table 2.6.

Operating System (OS) Level Virtualization (Containers)

The Platform as a Service (PaaS) model has accelerated application development and eliminated the need for administration of the underlying infrastructure. In this service model, application isolation is achieved through the utilization of containers that can run both on PMs and VMs.

Containers are the building blocks of OS-level virtualization that offer isolated virtual environments without the need for intermediate monitoring media such as hypervisors, as shown in Figure 2.2. The container technology of the Linux Kernel are developed separately by four different resources including OpenVZ [3] from Parallels, Google's cgroups (control groups), IBM's Dpar, and namespaces [139]. Among those, cgroups and namespaces presented solutions for resource management and per process isolation respectively and except for the Dpar, the other three are currently used [20].

Containerization technology has been implemented on large scale by cloud companies such as Google and Facebook. Containers are beneficial for cloud providers since they can be more densely packed when compared to VMs. The other benefit of containers is that they all share the host kernel. Therefore, the communication between containers and the hardware is performed through standard systems calls, which is much faster than hypervisor-based communication.

[3] https://openvz.org/Main_Page

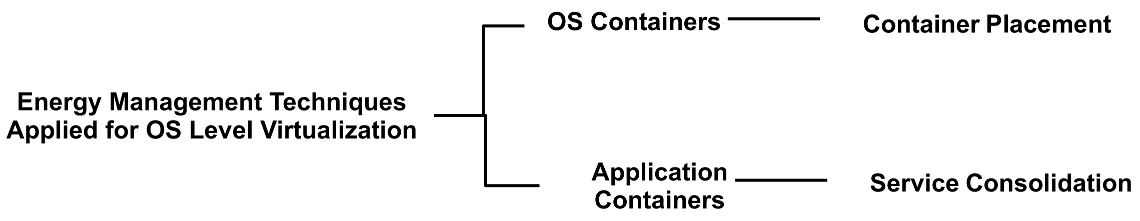


Figure 2.3: Energy management techniques which are applied to the OS level virtualization environments.

Operating System level virtualization or the containerization itself is categorized in two different types including OS containers and application containers and the energy management techniques which are applied to these environments are depicted in Figure 2.3. OS containers can be taught of VMs that share the kernel of the hosts operating system while providing isolated user space. Various OS containers with identical or different distributions can run together on top of the host operating system as long as they are compatible with the host kernel. The shared kernel improves the utilization of resources by the containers and decreases the overhead of container's startup and shutdown.

OS containers are built up on the cgroups and namespaces, whereas application containers are built upon the existing container technologies. Application containers are specifically designed for running one process per container. Therefore, one container is assigned for each component of the application. Application containers, as demonstrated in Figure 2.4, are specifically beneficial for microservice architecture in which the objective is having a distributed and multi component system that is easier to manage if anything goes wrong.

Operating System (OS) Containers

OS containers based on cgroups and namespaces provide user space isolation while sharing the kernel of the host operating system. The development in OS containers is like VMs and one can install and run applications in these containers as he runs it on a VM. Like VMs, containers are created from templates that identify the contents [121]. Google cluster is an example of such systems that runs all its services in containers. As stated

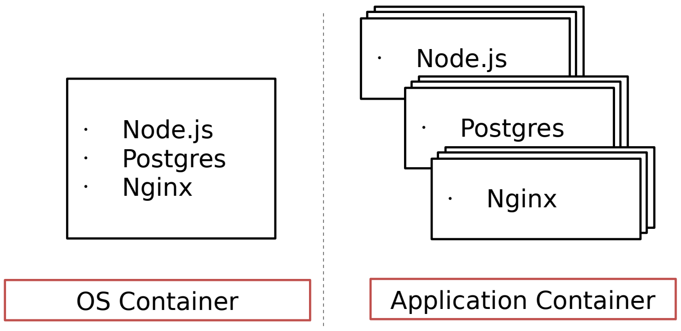


Figure 2.4: The differences between the Application container and the OS container for a three tier application. Application containers are implemented to run a single service and by default has layered Filesystems [121].

on Google open source blog [4], Google launches more than 2 billion containers per week considering all of its data centers. The container technologies that support OS containers are LXC [5], OpenVZ, Linux VServer[6], FreeBSD Jails and Oracle's Solaris zones [135].

Energy efficient resource management techniques applied for OS container systems mostly focus on the algorithms for initial placement of the OS containers. In this respect, Dong et al. [45] proposed a greedy OS container placement scheme, the most efficient server first or MESF, that allocates containers to the most energy efficient machines first. For each container, the most energy efficient machine is the server that shows the least rise in its energy consumption while hosting the container. Simulation results using an actual set of Google cluster data as task input and machine set show that the proposed MESF scheme can significantly improve the energy consumption as compared to the Least Allocated Server First (LASF) and random scheduling schemes. In addition, a new perspective on evaluating the energy consumption of a cloud data center is provided considering resource requirement of tasks along with task deadlines and servers' energy profiles.

Pandit et al. [126] also explored the problem of efficient resource allocation focusing on the initial placement of containers. The problem is modeled utilizing a variation of multi-dimensional bin packing. CPU, memory, network and storage of PMs are all considered as each dimension of the problem. In a general n-dimensional bin-packing problem, there exists n sub-bins of different sizes that must be filled with objects. The

[4] `http://google-opensource.blogspot.de/2014/06/an-update-on-container-support-on.html`
[5] `https://linuxcontainers.org/`
[6] `http://linux-vserver.org/Overview`

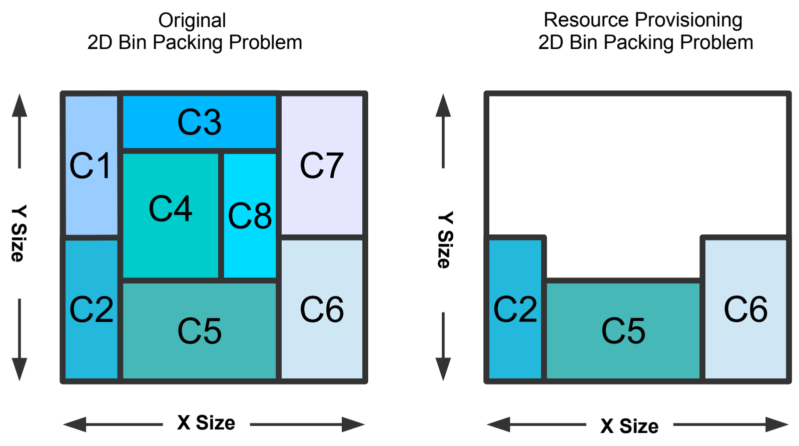


Figure 2.5: The difference between the original bin packing problem and its variation for the resource allocation [126]

resource allocation problem is different from the general form, since if any sub-bin of a bin reaches its capacity (e.g. CPU), then the bin is considered full while in the original problem this is not the case. Figure 2.5 demonstrates this difference. In order to design an efficient resource allocation algorithm, Pandit et al. [126] applied Simulated annealing (SA). SA is a technique used to find optimal or sub-optimal solution for NP Hard problems such as the bin packing problem and it is often applied for discrete search space. The proposed resource allocation algorithm is demonstrated to be more efficient in terms of resource utilization when compared to the commonly used First Come First Serve (FCFS) allocation policy.

OS containers is also utilized in Mesos [81] to provide the required isolation for workload. Mesos platform enables sharing commodity clusters between cluster computing frameworks with different programming models. The main objective of Mesos is efficient utilization of resources through sharing and also avoiding data replication for each framework. Hindman et al. [81] proposed a two-level scheduling for the Mesos platform called 'resource offers'. For the first level of the scheduling, Mesos identifies the amount of required resources for each framework. The second level scheduling is performed by the scheduler of each framework, therefore the scheduler has the ability to accept or reject the resources while deciding about the placement of the tasks. Mesos used Linux Containers and Solaris technologies for the workload isolation. The framework is tested through applying both CPU and IO-intensive workloads derived from the statistics of

Facebook cloud backend traces. The studied workloads are derived from the applications that are developed utilizing both Hadoop and MPI programming model. Results show that Mesos is highly scalable and fault tolerant and can improve the resource utilization with less than 4% overhead.

Application Containers

Contrary to OS containers that run multiple processes and services, application containers are dedicated to a single process and are built upon OS containers. The single process in each container is the process that runs the residing application [121]. Application containers can be considered a new revolution in the cloud era since containers are lightweight, easier to configure and manage, and can decrease the start-up time considerably. Docker [7] and Rocket [8] are examples of application containers. These containers are the building block of modern PaaS. Regular provisioning and de-provisioning of these containers, that happens during the auto-scaling, along with their unpredictable workloads results in cloud resource wastage and consequently more energy consumption. Therefore, like OS containers, designing optimal placement algorithms is the major challenge for container-based cloud providers.

Containers are fast to deploy because of their low overhead. Therefore, to simplify the development of applications, Spicuglia et al. [147] proposed OptiCA in which applications execute inside containers. The aim of the proposed approach is to achieve the desired performance for any given power and capacity constraints of each processor core. Although the focus in OptiCA is mainly on effective resource sharing across containers under resource constraints, it still reduces power consumption through considering energy as one of the constraints.

Anselmi et.al [5] investigated the Service Consolidation Problem (SCP) for multi-tier applications with the objective of minimizing the number of required servers while satisfying the Quality of Service defined by applications' response time. For modeling the data center, queueing networks theory is utilized since it is capable of capturing the performance behavior of service systems. A number of linear and non-linear optimization

[7] `https://www.docker.com/`
[8] `https://rocket.readthedocs.org/en/latest/`

server consolidation problems are defined and solved through a number of heuristics. Heuristics are chosen as they solve the optimization problems in a shorter amount of time with a considerable accuracy when compared to the standard Integer Linear Programming (ILP) techniques. This work solves SCP and finds the best data-center configuration with the least cost while satisfying the required end-to-end response time of applications.

In the same direction, Rolia et al. [138] investigated the SCP problem with the objective of minimizing the number of required servers through application consolidation. Enterprise applications are studied and their resource utilization is characterized. Like Anselmi et al. [5], linear integer programming is considered as one of the solutions and ILP is further compared with the genetic algorithms. The techniques are validated through a case study considering the workload of 41 servers. Results show that the linear integer programming model outperforms the genetic algorithm in terms of the required computation with a satisfactory accuracy in estimating the resource utilization. However, the proposed technique is not evaluated for large-scale data centers containing thousands of servers.

Mohan Raj et al. [116] also focused on minimizing the energy consumption of the data center through consolidation of applications on physical machines (PM). An end-to-end Service-Level Agreement (SLA)-aware energy efficient strategy is presented in [116] and the main objective is having an strategic workload scheduling for maximizing energy efficiency of the data center. Tasks are consolidated on virtual machines so that the number of active servers is reduced. Contrary to the previous discussed works [5, 138], synthetic workloads following the Poisson distribution are applied for the simulations to model the web server workloads. Containers are placed on the PMs that utilize the least energy rise. SLA is maintained through a control theoretic method and the requests of applications are accepted considering the SLA along with the data center capacity. The presented model is a queue-based routing approach and Holt-Winters forecasting formula is utilized for improving the SLA through decreasing the cost incurred by the times system waits for a PM to startup or to shut down. The proposed algorithm is also applicable in virtualized environments, where applications execute on virtual machines instead of directly on PMs.

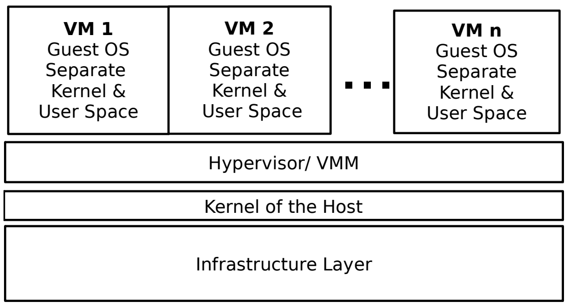


Figure 2.6: System Level Virtualization

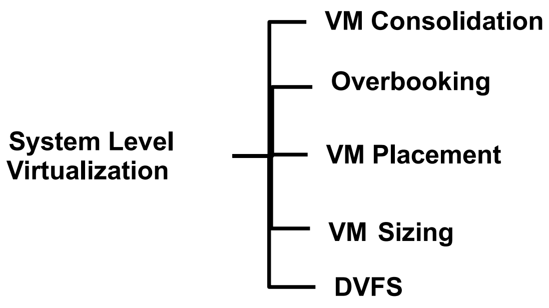


Figure 2.7: System-Level virtualization energy efficient management techniques

2.2.3 System-Level Virtualization (Virtual Machines)

The virtual machine's idea originated from simulated environments offered for software development when testing on real hardware was unfeasible [68]. In this respect, a specific environment was simulated considering the required processor, memory, and I/O devices. Later, this idea was improved to develop efficient simulators to provide copies of a server on itself. The improvement is done so that the program running on each copy can be executed directly on the hardware without requiring any software interpretation. These copies (simulated environments) are referred to as virtual machine systems, and the simulated software is referred as the virtual machine monitor (manager) (VMM) or the hypervisor [68]. This kind of virtualization is referred as system-level virtualization.

In system-level virtualization, the VM communications happens through the hypervisor with more overhead than the OS standard calls for containers. However, communicating through VMM offers a stronger layer of isolation and is more secure than containers [121]. In addition, system-level virtualization enables VMs with any type of OS to be installed on a host (Figure 2.6). Moreover, this technology enables consolidating virtual machines (VM) on physical machines (PMs) to reach higher and efficient utilization

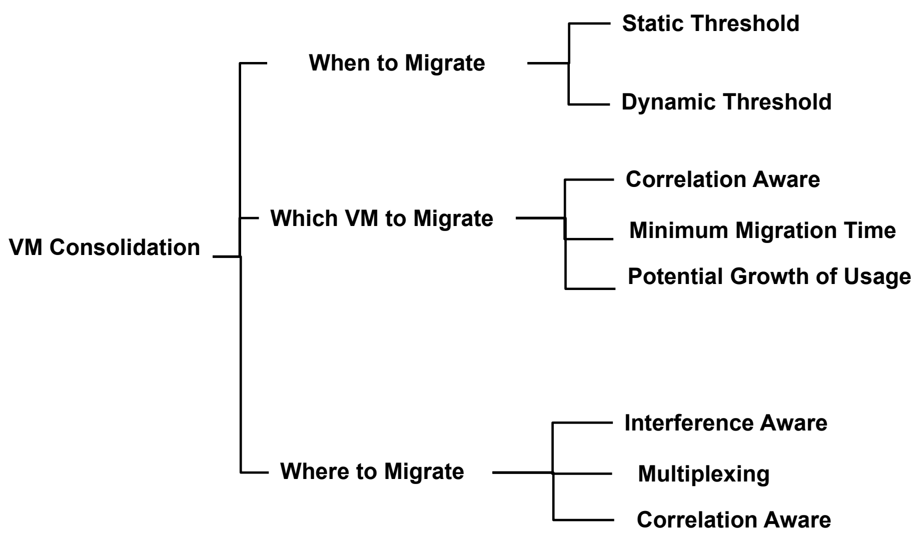


Figure 2.8: The consolidation sub problems which need to be answered for a general consolidation problem.

levels of PMs. The virtualization technology also improves the deployment time, and the operating costs. As depicted in Figure 2.7, energy management technique applied for system-level virtualization are categorized into five groups namely virtual machine consolidation, overbooking, VM placement, VM sizing, and DVFS. The research in this area are summarized in Tables 2.7 - 2.9. In the rest of this section, we discuss these techniques with more details.

VM Consolidation

The hypervisor technology enables consolidation of virtual machines on physical servers. There is a vast body of literature investigating VM consolidation algorithms that can improve the energy consumption of data centers. The consolidation problem can be divided into three main sub-problems which are depicted in Figure 2.8. The techniques are grouped according to the sub problem it investigates.

Techniques investigating migration triggers (When to migrate?)

Virtual machine consolidation is shown to be an effective way to minimize the energy consumption of cloud data centers. However, identifying the right time to trigger migration is crucial specially when the host is overloaded. This ensures a certain level of Quality of Service (QoS).

Gmach et al. [67] proposed an energy-efficient reactive migration controller that identifies situations in which the hosts are determined overloaded or underloaded. The overload and underload detection is defined when the server's CPU and memory utilization goes beyond or under a given fix threshold respectively. The same approach is applied in [13] and the effect of these two thresholds on the overall data center energy consumption and SLA violations is studied. 30% and 70% is shown to be the efficient underload and overload threshold considering the total energy consumption and average SLA violations. Contrary to Gmach et al. [67], the proposed approach [13] is not dependent on the type of workload.

Beloglazov et al. [14] improved the aforementioned approach [13] so that the underload and over-load thresholds are automatically adjusted. The previous approach for triggering the migration is modified since fixed values for thresholds are not suitable for cloud environments in which the workload's behavior is unknown and dynamic. The automation is performed through statistical analysis of the historical data from virtual machines workload. CPU utilization of the host is assumed to follow the t-distribution so that the sample mean and the standard deviation of the distribution can be used for determining the overload thresholds of each host. However, only one underload threshold is defined for the whole system. The adoptive approach shows a considerable improvement in terms of the QoS when compared to the fixed thresholds while it still saves energy.

Cloud providers should be able to ensure the Quality of Service (QoS) that they have promised to costumers. In the consolidation process, this QoS might be degraded because of the hosts being overloaded. In this respect, Beloglazov et al. [15] proposed a technique for host overload detection that ensures QoS while saving energy. The proposed approach can find the optimal solution for the overload detection problem considering any known stationary workload. The main objective is maximizing the intermigration time considering a given QoS goal based on a Markov chain model. In order to handle the nonstationary workloads which are unknown, a heuristic-based approach is presented that utilizes the Multisize Sliding Window technique for workload estimation. The algorithm is validated through simulations considering PlanetLab VMs traces as the input workload. The technique is proven to provide up to 88% of the performance of the optimal offline algorithm.

Techniques for choosing VMs to migrate (What to migrate?)

When migration is triggered, the second step is selecting the appropriate virtual machine to migrate. For under-load hosts, it is clear that all the VMs should be migrated so that the host can be shutdown or put in a lower power state. For overloaded hosts only a couple of VMs are needed to be migrated so that the host is no longer overloaded.

Beloglazov et al. [15], investigated the problem of VM selection policies in the consolidation process. Three different VM selection algorithms are studied namely random selection (RS), maximum correlation (MC), and the Minimum Migration Time (MMT) policies. The RS policy chooses VMs randomly until the host is not overloaded anymore. The MC policy chooses the VM with the maximum correlated workload with the other co-located VMs. The MMT policy selects the VM with the least migration time. The performances of these policies are validated through simulation and the VM types are derived from Amazon EC2 instance types[9]. PlanetLab's workload[10] is used as the CPU utilization of the VMs. Since the applied workload is for single core VMs, the VM types are all assumed to be single-core and the other resources are normalized accordingly. The performance of the algorithms is compared considering the total energy consumption by the physical servers of the data center and the SLA violations. Considering the results, the MMT selection policy outperforms the MC and RS policies in terms of the total joint power consumption and the SLA violations. Minimization of the VM migration time is proven to be more important than the correlation between the VMs allocated to a host.

Beloglazov et al. [14] also compared two other VM selection policies including Minimization of Migrations (MM) and Highest Potential Growth (HPG) with the RS. The MM algorithm selects the least number of VMs to migrate with the objective of decreasing the migration overhead. HPG chooses VMs with the lowest CPU usage compared to their requested amount, aiming at reducing the SLA violations through avoiding the total potential increase. It is shown that MM, which reduces the number of migrations, outperforms the other two algorithms in terms of the SLA violations and data center energy consumption.

Techniques investigating migration destination (Where to migrate?)

[9]`https://aws.amazon.com/ec2/instance-types/`

[10]`https://www.planet-lab.org/`

When the migration is triggered and the virtual machines are selected for migration, it is the time to find a new destination for the selected VMs. Here, we describe techniques considered for finding new placement/destination for migrating virtual machines.

Interference-aware VM placement algorithms

Virtualization improves resource utilization efficiency and consequently the energy consumption of cloud data centers by enabling multi-tenant environments in which diverse workload types can exist together. VM consolidation and resource overbooking improve energy savings in cloud environments. However, overbooking might affect the performance of virtual machines that are co-located on each server VM [122]. The high-competitions for resource incurred between co-hosted VMs and the resource sharing nature of virtualized environment might cause performance degradation and more energy consumption. The degradation effect of co-located VMs on the performance of each others applications on the same VM is known as **performance interference** phenomenon.

Moreno et al. [119] investigated the impact of this phenomena on the energy efficiency in cloud data centers. The problem is formulated for the virtual machine placement and is modeled utilizing the Google cloud backend traces to leverage cloud workload heterogeneity. Google tasks are grouped according to their CPU, memory and length. Three types referred as small, medium and large are extracted through applying K-means clustering algorithm on the 18th day of the trace that has the highest submission rate. The VM placement decision is made considering the current performance interference level of each server. The interference aware VM placement algorithm is compared , via simulation, with the Google FCFS algorithm and shown to reduce the interference by almost 27.5% while saving around 15% of energy consumption.

As mentioned previously, the performance interference might degrade the QoS for real time cloud applications and consequently result in SLA violations and the placement/packing of VMs play an important role on the performance interference, since it is dependent on the workload of the co-located VMs. In this respect, Cagar et al. [23] presented an online VM placement technique included in the hALT (harmony of Living Together) middleware. hALT takes into account workload characteristics of the VMs along with the performance interference for finding placement for each VM. Machine Learning techniques are used for the online placement and the system is trained utilizing the results from an offline workload characterization. The presented framework contains three main parts, namely Virtual machine classifier, neural networks, and the decision making placement. A brief analysis of the Google cloud backend traces is presented. The analysis of the 3 days of the traces is utilized for training the classifier.

Each task in the Google traces is considered as a VM. CPU utilization, memory, and the CPI of tasks are used for the classification purpose. CPI attribute shows the 'Cycle Per Instruction' metric and is used as a performance metric since it can well present the response time for compute-intensive applications [172]. Therefore, tasks that utilize more than 25% of the CPU and are compute intensive are considered for evaluation of the framework. Decrease CPI results in better performance, a result that had been demonstrated by the previous study.

Back propagation-based artificial neural networks (ANN) [79] is used as the classifier that predicts the performance interference level. This is used to determine the best placement of the VM. The ANN is trained using the VM utilization patterns of each class of VMs, which is defined by the k-means clustering algorithm. The number of VM classes is estimated based on the maximum Silhouette value [92, 140] and is determined to be 6 for the studied data set. The effect of the performance interference on the energy consumption is not investigated. The other drawback of the work is when the VM migration is triggered the ANN should run for every server in the data center, which may cause delays and overhead for large data centers. This can be avoided through new techniques in the search process.

Multiplexing placement algorithms

The other technique that is widely applied to find the new placement for selected VMs is Statistical Multiplexing. Multiplexing means sharing a resource between users with the objective of increasing the bandwidth utilization. This method has been applied to a variety of concepts including MPEG transport stream for digital TV [77], UDP and TCP [58] protocols.

Meng et.al [113] applied the Statistical Multiplexing concept to VM placement in the server resources are multiplexed to host more VMs. In the proposed approach, called joint-VM provisioning, multiple VMs are consolidated together according to their workload patterns. Statistical multiplexing enables the VM to borrow resources from its co-allocated VMs while it is experiencing its workload spikes. In order to satisfy QoS requirements, a performance constraint is defined for each VM. This constraint ensures the required capacity for a VM to satisfy a specific level of performance for its hosted application. Three different policies are proposed for defining the performance constraint, selecting the co-located VMs, and estimating the aggregated resource demand of multiplexed VMs with *complementary* workloads. The VM workloads of a commercial data center are applied for evaluation purposes and the results show that the joint-VM provisioning is considerably efficient in terms of the energy consumption.

Chen et al. [29] investigated the problem of VM placement with the focus on consolidating more VMs on servers. The VM placement is formulated as a stochastic bin packing problem and VMs are packed according to their effective size (**ES**), which is defined considering Statistical Multiplexing principles. This principle takes into account the factors that might affect the servers aggregated resource demand on which the VM is placed. The effective size is originated from the idea of effective bandwidth, however it is extended to consider the correlation of VM workloads. In this respect, the ES of a VM is affected by its own demand and its co-located VMs considering the correlation coefficient. The proposed VM placement algorithm with the order of O(1) is applied to find the best destination for VMs. Poisson and normal distributions are considered for the VM workloads. The system is also validated through simulation applying a real cloud workload trace. The effective sizing technique adds around 10% to 23% more energy saving than a generic consolidation algorithm. The optimization is performed considering only one dimension, which is CPU demand of VMs.

Correlation Aware VM Placement

Verma et al. [159] are among the first researchers to take into account correlation between workloads of co-allocated VMs in the proposed consolidation approach. The idea is initiated from a detail study of an enterprise server workload, the distribution of the utilization and spikes of the workload while considering workloads statistic metrics including the percentiles and average. According to the analysis, average is not a suitable candidate for sizing the applications since the tail of the distribution of the utilization does not decay quickly for most of the studied servers. Therefore, if the sizing is performed based on the average, it might result in QoS degradation. However, the

90-percentile and the cross correlation is shown to be fairly stable and consequently are the best metrics for application sizing purpose. Two correlation-aware placement algorithms, Correlation Based Placement (CBP) and Peak Clustering based Placement (PCP), are proposed considering the insights from the workload characterization. The placement algorithms are implemented as a part of a consolidation planning tool and further evaluated utilizing traces from a live production data center. PCP achieves more energy savings than PCB and also improves the QoS through an extra metric to ensure that co-allocated workloads' peak do not lead to violations.

Similarly, Meng et al. [113] also utilized correlation of VMs in their proposed joint-VM provisioning approach. In this approach, multiple VMs are consolidated in a way that the underutilized resources of one VM can be used by the co-located VM at its peak.

Quality of Service (QoS) is important in a cloud environment especially for scale-out applications [54] such as MapReduce[36] and web searches. Therefore, Kim et.al [97] investigated the VM consolidation problem concentrating on the aforementioned applications. Scale-out applications are different from HPC workloads in terms of the workload variance resulted from their user-interactive nature. This high variance is caused by external factors such as number of users and makes these workloads less predictable. The other difference is that scale-out applications are latency sensitive and should maintain users expectations and consequently satisfy the SLA. In order to save power consumption, DVFS is considered in conjunction with the VM consolidation. The voltage and the frequency are defined considering the correlation between the VMs' workloads which ensures that the expected QoS is achieved. The approach is verified through real implementation of distributed web search applications. The approach is also validated for large scale cloud workloads and compared with a correlation aware placement approach [159]. The comparison shows that this approach outperforms the aforementioned technique by 13.7% and 15.6% in terms of the energy saving and QoS improvements.

Overbooking

Overbooking is an admission control method to improve resource utilization in cloud data centers. In general, cloud users overestimate the VM size they need to avoid risk of resource shortage. This provides the opportunity for providers to include an overbooking strategy [153] in their admission control system to accept a new user based on anticipated resource utilization and not on the requested amount. Overbooking strategies mostly rely on load prediction techniques and manage the tradeoff between maximizing resource utilization and minimizing performance degradation and SLA violation.

Based on the resources considered for overbooking, research works can be classified into two main categories. The first category [78, 96] only considers CPU and the second category [153, 154] considers I/O and memory along with CPU. Commonly, after the overbooking phase, the majority of approaches [84, 150] mitigate the risk of overbooking by dealing with overload of VMs on a limited number of servers. However, when the data center is overloaded, such techniques are no longer effective.

One way to deal with such challenges (which is of interest for PaaS provider) is to collect the statistics regarding application performance metrics and then, based on the priority of application and users, degrade user experience and reduce utilization of resources. There are a number of application-aware approaches proposed in the literature [78, 96]. However, they are application-specific and only consider CPU. To this end, Klein et al. proposed brownout [101], a programming paradigm that suits cloud environments and considers CPU, IO and memory. In a PaaS environment, brownout is integrated to an application in three phases. In the first phase, the application owner (with the incentive of receiving discount on service cost) reveals which part of the hosted application can be considered non-compulsory. This part of application can be discarded to decrease the resource requirements. In the second phase, brownout decides how often the non-compulsory computation can be discarded. Finally, in the last stage, when there is not enough capacity, brownout lessens the number of requests served with the non-compulsory part.

VM Placement

Among resource management policies, the initial placement of VMs plays an important role in the overall data center performance and energy consumption. A strategic placement of VMs can further improve the system overhead through decreasing the required number of migrations. Kabir et al. [88] investigated the issue assuming a hierarchical structure as a favored deployment model for cloud service provider that consists of cloud, cluster, and hosts. This model helps in appropriately managing the geographical distributed infrastructure to achieve scalability. This hierarchical structure needs a VM placement approach that smartly provides cloud cluster, and node selection mechanisms to minimize resource fragmentation and improve energy efficiency.

In general, the placement strategies can be categorized into two classes, namely centralized and hierarchical. Khosravi et al. [95] proposed a centralized VM placement algorithm for distributed cloud data centers with the objective of minimizing both power consumption and carbon footprint. An information system that has the updated status regarding cloud, cluster, and host utilization is considered that enables centralized decision making and resource optimization. They considered distributed data centers with diverse carbon footprint rates and PUE values and provided a comprehensive comparison on energy efficiency of different combinations of bin-packing heuristics. They concluded that the proposed approach called energy and carbon-efficient (ECE) VM placement saves up to 45% carbon footprint and 20% of power consumption in data centers.

Similarly, Forestiero et al. [57] proposed EcoMultiCloud, a hierarchical approach for workload management that offers an energy efficient VM placement in a multi-site data center. Their proposed architecture consists of two main layers. The upper layer is responsible for the assignment of workload (virtual machine requests) among remote sites and lower layer places virtual machines to hosts in each site. The proposed hierarchical approach achieves same energy efficiency as ECE (centralized solution), and offers more flexibility. This is because, as a hierarchical approach, it allows single data centers to select their internal VM placement algorithms.

VM Sizing

Virtualization technology provides the opportunity for applications to share the underlying hardware with secure isolation [113]. Virtual machines configuration, in terms of the amount of resources (CPU, memory and I/O), are pre-defined by the cloud provider in most of the cloud service models. VM configuration is important for the resource allocation process where a host with enough resources need to be chosen to host the VM. Such VM placement process may ultimately affect the energy consumption of the data center. Therefore, the efficiency of VM placement can be achieved by three different ap-

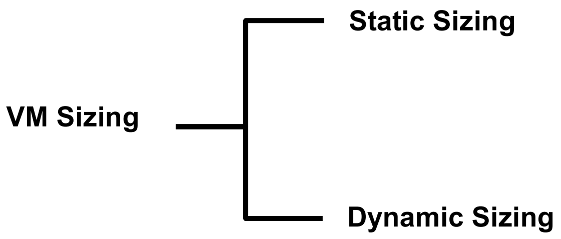


Figure 2.9: VM sizing techniques categorized in two major groups including static and dynamic sizing.

proaches. In the first approach, VMs are assigned to hosts according to their fix sizes and consolidated to less number of servers without change of configuration. This approach is discussed with details in the VM consolidation section [13–15, 67, 119]. The second approach is tailoring virtual machine configuration to the workload, which can be achieved through characterization of the applications workload. These two approaches are considered as *Static VM Sizing* (Figure 2.9). Finally, the third approach is adjusting the VM's configuration to match its workload in runtime [113] and is known as *Dynamic VM Sizing* (Figure 2.9).

Static VM Sizing

Assuncao et al. [8] proposed CloudAffinity, a framework to match physical servers to VM instances called as CloudMates. This framework enables organizations to move their workloads to the cloud while choosing optimal number of available VM templates considering their budget constraint. CloudAffinity considers CPU, memory, and disk requirement of each server and chooses the optimal number of VM templates minimizing the users cost based on the predefined Quality of Service (QoS). The QoS is defined as the percentage of the requests which are satisfied by each VM template. The effectiveness of the VM template matching is investigated through three metrics including cost, Euclidean distance, and Matching factor. The cost metric shows the amount of money that the user should pay to maintain a cloud instance and this cost differs from one instance to the other. The Euclidean distance metric is the distance between the cloud providers template and the users requirement in terms of the resources including CPU, memory and disk. The Matching factor metric shows the percentage of the difference between customers requirement and what the template offers for each VM.

Dynamic VM Sizing

In dynamic VM sizing, the approach estimates the amount of the resources that should be allocated to a VM with the objective of matching the VMs resources to its workload. The VM sizing should be carried out in a way that to avoid SLA violations resulted from under-provisioning of resources. Meng et al. [113], present a provisioning approach in which the less correlated VMs are packed together and an estimate of their aggregate capacity requirements is predicted. This approach applies statistical multiplexing considering dynamic VM resource utilization and it makes sure that the utilization peaks of one VM do not necessarily coincide with the co-allocated VMs. Hence, the amount of resources allocated to each VM varies according to its workload.

Chen et al. [29] also investigated the problem of VM sizing in the provisioning process, so that more VMs can be hosted on servers. The estimated VM sizes are referred as effective size (**ES**), which is determined through Statistical Multiplexing principles. These principles take into account the factors that might affect aggregated resource demand of the server on which the VM is placed. The effective size of a VM is affected by both its own demand and its co-allocated VMs considering the correlation coefficient. This effective sizing technique is demonstrated to save more energy than a generic consolidation algorithm.

Dynamic Voltage and Frequency Scaling (DVFS)

In addition to the *Bare Metal* environment, Dynamic Voltage Scaling has also been applied to virtualized environments. Laszewski et al. [160] focused on the design and implementation of energy-efficient VM scheduling in a DVFS-enabled compute clusters. Jobs are assigned to preconfigured VMs and the VMs are shutdown when the jobs finish. The solution is proposed for high performance cluster computing, however since they consider virtualization technology, it can be implemented in a cloud environment as well. The scheduling algorithm operates considering two main approaches. It either optimizes the processor power dissipating by running the CPU at a lower frequency with the minimum effect on the overall performances of the VMs or schedules the VMs on CPUs with low voltages and tries not to scale up the CPU voltage.

Urgaonkar et al. [156] investigated the problem of optimal resource allocation and

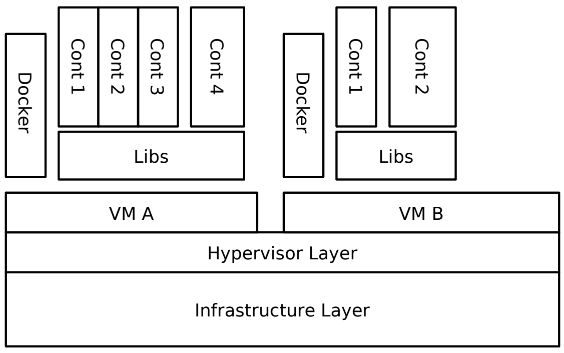


Figure 2.10: Hybrid Virtual Environment

power efficiency in cloud data centers through online control decisions. These decisions are made utilizing available queueing information in the system and the Lyapunov optimization theory. Heterogeneous applications with volatile workloads are used to show that the presented approach can handle unpredictable changes in the workload since it is not dependent on any prediction or estimate of the workload. In the studied system, applications execute inside virtual machines and each application can have multiple instances running across different VMs. Dynamic Voltage and Frequency scaling of CPU is applied for improving the energy consumption of the data center. The frequency of CPU is decided according to the workload by the *Resource Controller*, which is installed on each server.

Authors conclude that DVFS does not always result in more energy savings and operators should also consider utilizing the low power modes available in modern processors which might provide better energy savings with the least performance degradation.

2.2.4 Hybrid

The hybrid virtualization model shown in Figure 2.10 is a combination of system and OS-level virtualization approaches. The containerization technology used in this model is mainly application containers such as Docker [11] , which was explained previously. This new model is currently provided by Google Container Engine and Amazon ECS as a new cloud computing service called Container as a Service. Running containers inside virtual machines provides an extra layer of isolation while ensuring the required security for

[11] https://www.docker.com/

users. Research in this area are summarized in Tables 2.10.

As discussed, VM consolidation can be utilized for reducing energy consumption in data centers. However, VM consolidation is limited by the VMs memory [151] and unlimited number of VMs can not be mapped on a physical machine (PM). Therefore, in order to save more energy, VM's resources should also be utilized efficiently. This problem is tackled in the hybrid model through consolidation of containers on VMs, which further improves the utilization of VMs resources. This approach is introduced by Tchana et al. [151] and is called *Software/Service Consolidation* problem (SCP). In the SCP problem, several software/services are dynamically collocated on one VM. The objective is reducing the number of VMs and consequently decreasing the population of PMs along with the data center power consumption. In order to provide the required isolation, applications execute inside Docker containers. The problem is modeled utilizing Constraint Satisfaction Programming (CSP) and the *Software Consolidation* is carried out along with the VM consolidation. In order to accelerate the software consolidation process, the search domain is reduced considering a couple of boundaries such as containers collocation constraints. One of the limitations of this approach is the OS of the VM hosting the container, since unlike virtual machines, containers share the OS with their host.

Yaqub et al. [168] also investigated SCP in PaaS Clouds. This problem is framed leveraging the Google definition of Machine Reassignment model for the ROADEF/EURO challenge [12] and was extended for RedHat's public PaaS (OpenShift[13]). Four Meta-heuristics are applied to find solutions for (re)allocations of containers. The solutions are then compared and ranked considering SLA violations, energy consumption, resource contention, migrations, machine used, and utilization metrics for four different cloud configurations. Contrary to [151], no boundaries are considered to reduce the search domain for the Meta-heuristics to speed up the consolidation process.

Almeida et al. [4] investigated the SCP problem in a Service-Oriented Architecture. The problem was divided into two related sub-problems, short-term resource allocation and long-term capacity planning. The short-term resource allocation problem has a short-term impact on the revenue and its solution determines the optimal resource allocation to different services while increasing the revenue obtained through SLA contracts. How-

[12]`http://challenge.roadef.org/2012/en/index.php`
[13]`https://github.com/openshift-quickstart`

ever, the answer to long-term problem determines the optimal size of the service center that maximizes the long-term revenue from SLA contracts along with decreasing the Total Cost of Ownership (TCO). These problems are modelled in the proposed framework and a deep analysis of effects of short-term resource allocation is provided. A model is presented for identification of the optimal resource allocation in order to maximize the revenues of the service provider while meeting the required QoS. Resource utilization and the associated costs are also taken into account. The proposed optimal model is fast in terms of the computation speed, which makes it a good candidate for online resource management. Transactional Web services are considered as the hosted applications in the data center. The services are categorized into sub-classes because of the volatility of the web server workloads. Each VM is responsible for one class of web servers (WS). In order to insure the quality of service for each class of the WS, admission control is employed on top of each VM which decides to accept or reject the requests.

Dhyani et al. [41], introduced a constraint programming approach for the SCP problem. The research objective is decreasing data center cost through hosting multiple services running in VMs on each host. The SCP is modeled as an Integer Linear Programming (ILP) problem and compared with the presented solution through constraint programming. The constraint programming approach can find the solution in less than 60 seconds. However, ILP could find a better solution if it could meet the 60 seconds deadline. Therefore, constraint programming is found as a better solution comparing to ILP for SCP problem considering the algorithm speed.

Bichler et al. [17] also investigated the problem of capacity planning. An IT service provider hosting services of multiple customers is investigated in a virtualized environment. Three capacity planning models are proposed for three allocation problems. These problems are solved through multi-dimensional bin-packing approximate algorithms and the workloads of 30 services are applied as the input of the system.

2.3 Workload Characterization and Modeling

There is a growing body of research on resource management techniques with the focus on minimizing the energy usage in cloud data centers [89,123]. These techniques should be applicable for dynamic cloud workloads. However, because of the competitiveness and security issues, cloud providers do not disclose their workloads, and as a result there are not many publicly available cloud back-end traces. Therefore, most of the research lacks the study of the dynamicity in users demand and workload variation.

The availability of cloud backend traces makes researchers able to model real cloud data center workloads. The obtained model can be applied for proving the applicability of the proposed heuristics in real world scenarios.

In 2009, Yahoo released traces from a production MapReduce M45 cluster to a selection of universities [167]. In the same year, Google made the first version of its traces publicly available and this publicity resulted in a variety of research investigating the problems of capacity planning and scheduling via workload characterisation and statistical analysis of the planet's largest cloud backend traces [137].

2.3.1 Workload Definition

The performance of a system is affected not only by its hardware and software components but also by the load it has to process [27]. As stated by Feitelson [51], understanding the workload is more important than designing new scheduling algorithms. If the tested systems does not have its input workload chosen correctly, the result of the proposed policies or algorithms might not work as expected when applied to real world scenarios.

The computer workload is defined as the amount of work allocated to the system that should be completed in a given time. A typical system workload consists of tasks and group of users who are submitting the requests to the data center. For example, in Google workload tasks are the building block of a job. In other words, a typical job consists of one or more tasks [137]. These jobs are submitted by the users, which are in this case the Google's engineers or its services.

2.3.2 Workload Modeling Techniques

In order to characterize the workload, the drive or input workload of the studied system should also be investigated. For measuring the performance of a computer system, input workload[14] should be the same as the real one. As stated by Ferrari [55], there are three types of techniques for obtaining the input workload:

- **Natural Technique**

  Natural technique utilizes real workloads obtained from the log file of the system without any manipulation. Urgaonkar et al. [156], utilized real traces from heterogeneous applications to investigate the problem of optimal resource allocation and power efficiency in cloud data centers. Anselmi et.al [5] also applied real workloads from 41 servers to validate their proposed approach for Service Consolidation Problem (SCP). PlanetLab VMs traces are applied as the input workload to validate the consolidation technique in several works [15, 21].

- **Artificial Technique**

  Artificial technique involves the design and application of a workload that is independent of the real one. Mohan Raj and Shriran [116] apply synthetic workloads following the Poisson distribution to model web server workloads.

- **Hybrid Technique**

  Hybrid technique involves sampling a real workload and constructing the test workload from the parts of the real workload. Hindman et al. [81] evaluate Mesos the application of both CPU and IO-intensive workloads that are derived from the statistics of Facebook cloud backend traces and running applications utilizing Hadoop and MPI.

Workload Modelling

As stated by Calzarossa and Swerazzi [27], the workload modeling process can be constructed through three main steps. The first step is the formulation in which the basic components such as submission rates for users and their descriptions are selected. In addition to this, for evaluating the proposed model, a criterion is considered. During the second step, the required parameters for modeling are collected while the workload

[14]The input or drive workload is the workload under which the performance of the system is tested.

executes in the system. Finally in the last step, a statistical analysis is performed on the collected data.

In selecting the workload modeling technique, the considered parameters for defining the requests play an important role [2]. In a distributed system, a user request is mainly defined via three main parameters including:

- t : The time t is when the request is submitted to the system.
- l : The location l is where the request is submitted from.
- r : The request vector r contains the amount of resources needed in terms of CPU, memory and disk.

When time and spatial distribution of the user requests are ignored, e.g., only one day of the trace is studied, requests population are likely to have similarities and can be presented in the form of relatively homogeneous classes [2]. Such kind of workload modeling is explored by Mishra et al. [115] and Chen et al. [31] on the first version of the Google cluster traces. Mishra et al. [115] applied the clustering algorithm K-means for forming the groups of tasks with more similarities in resource consumption and duration, while Chen et al. [31] classified jobs instead of tasks. In addition to these approaches, Di et al. [43] characterized applications running in the Google cluster. Like [31, 115], K-means is chosen for the clustering purpose.

If the time and location of the requests are considered, the workload can be modeled via a stochastic process such as Markovian model or time series models such as the technique applied by Khan et al. [94]. Khan et al. [94] presented an approach based on Hidden Markov Modeling (HMM) to characterize the temporal correlations in the clusters of VMs that are discovered and to predict the patterns of workload along with the probable spikes.

2.3.3 Workload-based Energy Saving Techniques

Study of the characteristics of the workload and its fluctuations is crucial for selecting energy management techniques. For example in Intel Enhanced Speed Stepping Technology[15], the CPU frequency and voltage are dynamically adjusted according to the servers

[15] http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm#overview

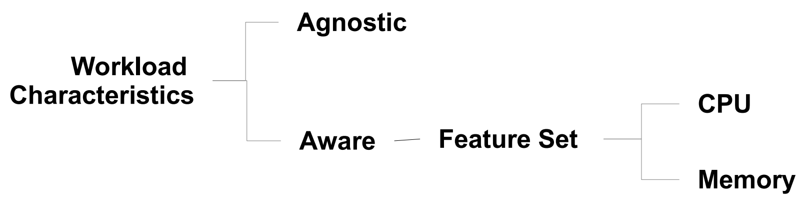


Figure 2.11: The energy efficient resource management techniques in PaaS environment are grouped based on the approach awareness of the cloud workload and its characteristics.

workload. From the analysis of the workload, one can decide if a power management methodology is applicable for the system. As stated by Dhiman et al. [40], DVFS does not always result in more energy savings and operators should also consider utilizing low power modes available in modern processors that might provide better energy savings with the least performance degradation considering the workload. The workload type is also important for DVFS on memory component because, as stated previously, in non-memory intensive workloads running at lower memory speed would result in less performance degradation than memory-intensive workloads. Therefore, reducing power consumption can be obtained through running memory at a lower frequency with the least effect on the application performance [35]. The energy efficient resource management techniques in PaaS environments are grouped into two major categories namely workload aware and workload agnostic as depicted in Figure 2.11.

Beloglazov et al. [15] applied Markov chain model for known stationary workloads while utilizing a heuristic-based approach for unknown and non-stationary workloads. Apart from this work, the analysis of workloads of co-existing/co-allocated VMs motivated new algorithms and management techniques for saving energy in cloud data centers. These techniques contain the interference-aware [23,119,122], and correlation-aware and multiplexing [29,54,113,159] VM placement algorithms, virtual machine static [8] and dynamic sizing techniques [113], which were discussed previously. The workload study also motivated the idea of overbooking resources to utilize the unused resources allocated to the VMs [84,150,153,154].

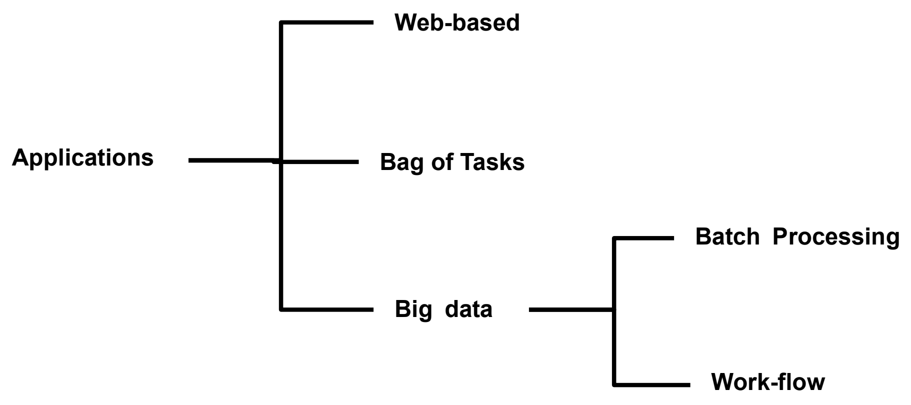


Figure 2.12: Application types supported in energy management systems.

2.4 Application-based Energy Saving Techniques

The type of application (Figure 2.12) plays an important role in selecting the energy management technique. For scale out applications, turning on/off cores, which is called dynamic power gating, is not practical since these applications are latency sensitive and their resource demand is volatile, therefore the transition delay between power modes would degrade the QoS. In this respect, Kim et al. [97] considered the number of cores according to the workloads peak and achieved power efficiency through DVFS.

2.4.1 Web Applications

Web applications deployed in cloud data centers have highly fluctuating workloads. Wang et al. [161] measured the impact of utilizing DVFS for multi-tier applications. They concluded that response time and throughput are considerably affected as results of bottlenecks between the database and application servers. The main challenge is identifying the DVFS adjustment period, which is not synchronized with workload burst cycles. Therefore, they proposed a workload-aware DVFS adjustment method that lessens the performance impact of DVFS when a cloud data center is highly utilized. VM consolidation methods also have been used along with DVFS for power optimization of multi-tier web applications.

Wang et al [162] proposed a performance-aware power optimization that combines either DVFS or VM consolidation. To achieve the maximum energy efficiency, they integrate feedback control with optimization strategies. The proposed approach operates in two levels: 1) at the application level, it uses a multi-input-multi-output controller to reach the performance stated in SLA by dynamically provisioning VMs, reallocating shared resources across VMs and DVFS, 2) at the data center level, it consolidates VMs onto the most energy-efficient host.

2.4.2 Bag of Tasks

Bag-of-Tasks (BoT) applications are defined as parallel applications whose tasks are independent [33]. Kim et al. [99] investigated the problem of power-aware BoT scheduling on DVS-enabled cluster systems. Applying DVFS capability of processors, the presented space-shared and time-shared scheduling algorithms both saved a considerable energy while meeting the user-defined deadline.

Calheiros et al. [24] proposed an algorithm for scheduling urgent, CPU intensive Bag of Tasks (BoT) utilizing processors DVFS with the objective of keeping the processor at the minimum frequency possible while meeting the user-defined deadline. An urgent application is defined as a High Performance Computing application that needs to be completed before the soft deadline defined by the user. Disaster management and healthcare applications are examples of this kind of applications. DVFS is applied at the middleware/Operating System level rather than at CPU level and maximum frequency levels are supplied by the algorithm during task execution. The approach does not require prior knowledge of the host workload for making decisions.

2.4.3 Big Data Applications

As indicated by International Data Corporation (IDC) in 2011, the overall information created and copied in the world has grown by nine times within five years reaching 1.8 zettabytes (1.8 trillion gigabytes) [61] and this trend would continue to at least double every two year. The exceptional growth in the amount of produced data introduced the phenomenon named Big Data. There exists various definitions for Big Data. However, Apache Hadoop definition is the one which is close to the concept of this study. Apache Hadoop defines Big Data as *datasets that could not be captured, managed and processes by general computers within an acceptable scope*. Big data analysis and processing along with the data storage and transmission require huge data centers that would eventually consume large amount of energy. In this respect, energy efficient power management techniques are really crucial for Big Data processing environments. In this section, we discuss batch processing and workflows as two examples of Big Data applications along with the techniques applied to make them more energy efficient (Figure 2.12).

Batch processing

Large-scale data analysis and batch processing are enabled utilizing data center resources through parallel and distributed processing frameworks such as MapReduce [36] and Hadoop [16]. The large scale data analysis performed by these frameworks requires many servers and this triggers the possibility of a considerable energy savings that can be obtained via resource management heuristics that minimize the required hardware.

As stated by Leverich et al. [107], MapReduce is widely used by various cloud providers such as Yahoo and Amazon. Google executes on average one hundred thousand MapReduce jobs every day on its clusters[17]. The vast usage of this programming model, along with its unique characteristics, requires further study to explore any possibilities and techniques that can improve energy consumption in such environments.

The energy saving in a cluster can either be made by limiting the number of active servers to the workload requirement and shutting down the idle servers or matching the compute and storage of each server to its workloads. Due to the special characteristics of the MapReduce frameworks, these options are not useful in these environments. Powering down idle servers is not applicable, as in MapReduce frameworks data is distributed and stored on the nodes to ensure reliability and availability of data. Therefore, shutting down a node would affect the performance of the system and the data availability even if the node is idle. Moreover, in a MapReduce environment, the mismatch between hardware and workload characteristics might also result in energy wastage (e.g. CPU idleness for I/O workloads). Also, recovery mechanisms applied for hardware/software failures increases energy wastage in MapReduce frameworks.

Leverich et al. [107] investigated the problem of energy consumption in Hadoop as a

[16]https://hadoop.apache.org/
[17]http://google-opensource.blogspot.de/2014/06/an-update-on-container-support-on.html

MapReduce style framework. Two improvements are applied to the Hadoop framework. Firstly, an energy controller is added that can communicate with the Hadoop framework. The second improvement is in the Hadoop data-layout and task distribution to enable more nodes to be switched off. The data-layout is modified so that at least one replica of a data block would be placed on a set of nodes referred as *covering Set* (CS). These *Covering Sets* ensure the availability of the data block when the other nodes that store the other replicas are all shutdown to save power. The number of replicas in a Hadoop framework is specified by users and is equal to three by default.

Lang et al. [104] proposed a solution called All-In Strategy (AIS) that utilizes the whole cluster for executing the workload and then power down all the nodes. Results show that the effectiveness of the algorithms directly depend on both complexity of the workloads and the time it takes for the nodes to change power states.

Kaushik et al. [93], presented GreenHDFS , an energy efficient and a highly scalable variant of the Hadoop Distribution File System (HDFS). GreenHDFS is based on the idea of energy-efficient data-placement through dividing servers into two major groups namely Hot and Cold zones. Data that are not accessed regularly are placed in the Cold zone so that a considerable amount of energy can be saved harnessing the idleness in this zone.

Long predictable, streaming I/O and parallelization and non-interactive performance are named as the characteristics of MapReduce workloads computations in Leverich et al. [107]. However, there exists MapReduce with interactive analysis (MIA) style workloads that have been widely used by organizations [30]. Since MapReduce makes storing and processing of large scale data a lot easier, data analysts are widely adopting MapReduce to process their data.

Typical energy saving solution obtained through maximization of server utilization is not applicable for MIA workloads because of two main reasons. Firstly, MIA workloads are dominated by human-initiated jobs that force the cluster to be configured to the peak load so that it can satisfy SLAs. Secondly, workload spikes are unpredictable and the environment is volatile because machines are added or removed from the cluster regularly. In this respect, Chen et al. [30] proposed BEEMR (Berkeley Energy Efficient MapReduce) as an energy efficient MapReduce workload manager inspired by an analysis of the Face-

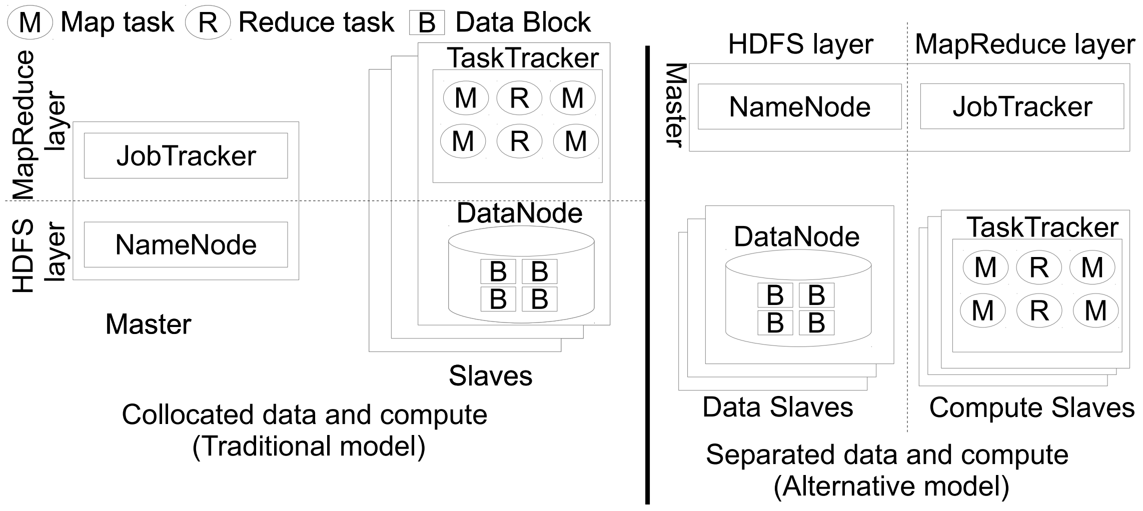


Figure 2.13: Two MapReduce development models studied in [52]

book Hadoop workload.

Hadoop is the open source implementation of the MapReduce programming model. Apart from energy consumption, which is studied in a number of works [30,93,104,107], Hadoop performance for both collocated and separated compute services and data models (Figure 2.13) is investigated by Feller et al. [52]. The separation of compute services and data is applied for virtualized environments. It is shown that the collocation of VMs on servers has a negative effect on the I/O throughput, which makes physical clusters more efficient in terms of the performance when compared to the virtualized clusters. The performance degradation is proven to be application-dependent and related to the data-to-compute ratio. There is also a tradeoff between the application's completion time and the energy consumed in the cluster.

Workflow Applications

Workflows or precedence-constrained parallel applications are a popular paradigm for modeling large applications that is widely used by scientists and engineers. Therefore, there has been an increasing effort to improve the performance of these applications through utilizing distributed resources of Clouds. With the increase in the interest toward this type of applications, the energy efficiency of the proposed approaches also comes into the picture, as performance efficiency brought by excessive use of resources might result in extra energy consumption.

The inefficiency of provisioned resources for scientific workflows execution results in excessive energy consumption. Lee et al. [106] addressed this issue through a resource-efficient workflow scheduling algorithm named MER. The proposed algorithm optimizes the resource usage of a workflow schedule generated by other scheduling algorithms. MER consolidates tasks that were previously scheduled and maximizes the resource utilization. Based on the trade-off between makespan (execution time) increase and resource utilization reduction, MER identifies the near optimal trade-off point between these two factors. Finding this point, the algorithm improves resource utilization and consequently reduces the provisioned resources and saves energy. The proposed algorithm can be applied to any environment in which scientific workflows of many precedence-constrained tasks are executed. However, MER is specifically designed for the IaaS cloud model.

As discussed earlier, Dynamic Voltage and Frequency Scaling (DVFS) is an effective approach to minimize the energy consumption of applications. As scientific workflows contain tasks with data dependencies between them, DVFS might not always result in desirable energy saving. Depending on system and workflow characteristics, decreasing the CPU frequency may increase the overall execution time and the idle time of the processors, which consequently deteriorates the planned energy saving. In addition, when the SLA violation penalty is higher than the power savings, adjusting the CPU to operate at the lowest frequency is not always energy efficient. In this situation, executing the tasks quickly with a higher frequency might result in less energy consumption [59]. In this respect, Pietri et al. [133] proposed an algorithm that identifies the best time to reduce the frequency in a way that the overall energy consumption is decreased. In the presented approach, the lowest possible frequency did not always result in the least energy consumption for completing the workflow execution. The algorithm considers various task runtime and processor frequency capabilities and it assumes an initial task placement on the available machines. Next, it determines the appropriate CPU frequency considering the time that the task can be stretched without violating the deadline (slack time). The proposed algorithm gradually scale down the frequency of the processor assigned for each task iteratively by the time the overall energy savings are increased. In each iteration, the CPU frequency is scaled down to the next available frequency mode. The al-

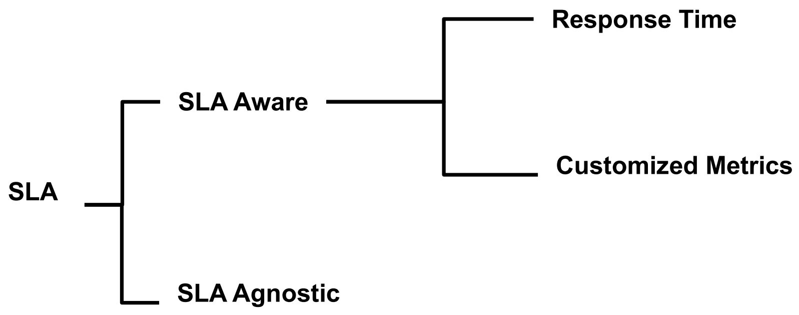


Figure 2.14: Considering SLA, the energy efficient resource management techniques for PaaS environments are categorized in two groups, namely SLA Aware and SLA Agnostic.

gorithm performance is validated through simulation and the results demonstrated that the system can provide a good balance between energy consumption and makespan.

Durillo et al. [46] proposed MOHEFT as an extension of the Heterogeneous Earliest Finish Time (HEFT) algorithm [155], which is widely applied for workflow scheduling. The proposed algorithm is able to compute a set of suboptimal solutions in a single run without any prior knowledge of the execution time of tasks. MOHEFT policy complements the HEFT scheduling algorithm through predicting task execution time based on the historical data obtained from real workflow task executions.

2.5 SLA and Energy Management Techniques

The expectations of providers and costumers of a cloud service including the penalties considered for violations are all documented in the Service Level Agreement (SLA) [72, 75, 169]. Considering SLA, energy management techniques are categorized into two groups, namely SLA-Aware and SLA-Agnostic approaches (as shown in Figure 2.14). SLA contains service level objectives (SLOs) including the service availability and performance in terms of the response time and throughput [100]. Satisfying SLA in a cloud computing environment is one of the key factors that builds trust between consumers and providers. There has always been a trade-off between saving energy and meeting SLA in resource management policies, therefore it is really crucial to make sure that energy saving does not increase SLA violations dramatically.

The metrics utilized to measure SLA can be different based on the application type, for example SLA for workflow applications is defined in terms of the user-defined dead-

Table 2.2: The thesis scope

Characteristic	Thesis scope
Virtualization	Containerized data centers
System resources	Multiple resources: CPU, RAM
Target systems	Heterogeneous Paas and CaaS Clouds
Goal	Minimize energy consumption under performance constraints
Power saving techniques	VM sizing, dynamic container consolidation, server power switching
Workload	Google cloud backend traces and Arbitrary mixed workloads
Architecture	Distributed task mapping and dynamic container consolidation systems

lines [46, 106, 133] while in web and scale-out applications it is defined as the response time [5, 81, 104]. Anselmi et.al [5] consider the application response time as their SLA metric in their proposed solution for the Service Consolidation Problem (SCP) considering multi-tier applications. In the studied scenario, the objective was minimizing the number of required servers while satisfying the Quality of Service. Similarly, Mohan et al. [116] considered response time of the application as the SLA metric in their proposed energy efficient workload scheduling algorithm. The application request is accepted considering the data center capacity along with the SLA. The SLA is maintained through a control theoretic method. Holt-Winters forecasting formula is applied for improving the SLA through minimizing the incurred cost by the time in which the system waits for startup and shutdown delays of a PM/VM. Cagar et al. [23] also considered response time as their SLA metric in the presented online VM placement technique. In a different approach, Beloglazov et al. [13] utilized a combined metric considering both SLA violation and energy consumption for their optimization problem. In the presented approach, SLA is violated during the time that a host is overloaded. This approach is application independent.

2.6 Thesis Scope and Positioning

This thesis investigates energy-efficient management of resources in enterprise and container-based clouds. The objective is to minimize energy consumption of data centers through efficient allocation of resources and consolidation of workload on minimum number of servers. The thesis investigates VM sizing, container placement and consolidation in a hybrid containerized environment and its scope is illustrated in Table 2.2. As inefficient utilization of servers is one of main source of the energy wastage in data centers[18], in this thesis we focus on improving servers utilization. We considered both OS-level and system-level virtualization technologies to improve utilization on VM and server level.

This thesis focus on Cloud environments where the tasks/applications are running inside containers. Chapter 3 focuses on VM sizing for containers while considering the analysis and application of real cloud backend traces. The effect of variable cloud workloads on the efficiency of VM sizing solutions has not been studied deeply in the literature. Hence, these chapters characterize the only publicly available cloud backend traces released by Google in 2011. Contrary to the literature that concentrates on the maximization of host-level utilization and load balancing techniques [89,98,123], our research in these chapters focus on maximizing the VM-level utilization via VM customization. These VM sizes are derived from the characterization of the Google workload via clustering techniques.

Chapters 4 and 5 studied energy efficient container placement and consolidation algorithms. To evaluate the performance of scheduling and allocation policies in containerized cloud data centers, there is a need for evaluation of environments that support scalable and repeatable experiments. In chapter 4, we introduce *ContainerCloudSim*, which provides support for modeling and simulation of containerized cloud computing environments. This simulator is proposed as the primarily focus of the current available simulators are solely on system level virtualization with virtual machine as the fundamental component [25,42,56,62,62,74,85,102,124,125,152,164] and do not support modeling and simulation of containers in a cloud environment.

Finally, Chapter 5 present a framework that consolidates containers on virtual machines. Our approach is different from the literature, as we modeled the consolidation problem on container level. We compare a number of container consolidation algorithms using the *ContainerCloudSim* simulator and evaluate their performance against metrics such as energy consumption and Service Level Agreement violations. We compare the energy efficiency of container consolidation with virtual machine consolidation to demonstrate the effectiveness of our approach. The proposed framework and algorithms can be applied to any containerized cloud environment to minimize energy con-

[18] `http://www.nrdc.org/energy/data-center-efficiency-assessment.asp`

sumption, or alternatively in a public cloud to minimize the total number of hours the virtual machines leased.

2.7 Summary

According to the Refrigerating and Air Conditioning Engineers (ASHRAE) [12], the Infrastructure and Energy Cost (I&E) has increased by 75% of the cost in 2014 while IT costs are only 25% [19]. This is a significant rise for I&E costs, which was contributing up to 20% to the whole cost when IT costs where only 80% in the early 90's. This drastic rise of data center power consumption has made energy management techniques a non-separable part of the resource management in a cloud computing environment. In this respect, there is a large body of literature that considers energy management techniques for various cloud service models. In this chapter, we mainly focused on the PaaS service model in which the data center owner can obtain prior knowledge of the applications and their usage patterns. Further, we discuss the energy management techniques in both bare-metal and virtualized environments. In summary, research in this area conclude that selecting the right energy management technique is dependent on three main factors:

- **The environment where the applications run.** In this chapter, we covered various alternatives for execution environments including Bare Metal, containerized, and hypervisor-based virtualization.
- **The workload and application type.** Applications are mainly different in terms of their workload patterns, latency sensitiveness, and etc. Understanding the workload characteristics can further improve the efficiency of the algorithms.
- **QoS.** The Quality of Service for applications is defined through the Service Level Agreements (SLA) and the SLA metric can be different considering the applications nature. Consideration of SLA is important since energy management techniques might result in SLA violations and consequently degrade the performance of the system or increase the total costs for the applications execution.

[19]http://www.electronics-cooling.com/2007/02/in-the-data-center-power-and-cooling-costs-more-than-the-it-equipment-it-supports/

Table 2.3: Energy Efficient Research Considering Bare Metal Environment

Authors	Workload	SLA	Energy Saving	Energy Model
David et al. [35]	SPEC CPU2006 benchmark workload	Application slow down	**DVFS** on Memory component for CPU intensive work-loads	Real-system measurements along with an analytically power reduction model for memory.
Deng et al. [39]	Memory Intensive, CPU Intensive, and balanced Workloads	User defined performance degradation limit.	**DVFS** on memory component	A power model is proposed to include the effect of the memory frequency on the systems power consumption.
Deng et al. [38]	Memory and CPU intensive and a combination of these workloads, The workload characteristics are known beforehand	Application performance degradation limit	**DVFS** on both CPU and mem-ory component.	System total energy model is proposed containing both CPU and memory frequency
Kim et al. [99]	Bag-of-Tasks	Task Execution time	**DVFS** on CPU component	$E = E_{dynamic} + E_{static} = k_1V^2L + k_2(k_1V^2L) = \alpha V^2L$
Leverich et al. [107]	Hadoop's MapReduce workload	Throughput	**Powering down idle nodes** which are not ac-cessed regularly, a set of nodes are considered as a backup to ensure the data availability.	Linear Power Model. (CPU only)
Lang et al. [104]	Workload Characteristics such as the expected resource consumption and performance goals of jobs are studied and considered as abstract meta-models	Response Time	**Power down/up MR nodes** to save energy in periods of low utilization.	An energy model is presented which incorporates the power drawn by both online and offline nodes during the workload execution.

Table 2.4: Energy Efficient Research Considering Bare Metal Environment

Authors	Workload	SLA	Energy Saving	Energy Model
Kaushik et al. [93]	one-month of Yahoo Hadoop's HDFS logs in a multi-tenant cluster are grouped according creation date and access rate. Two main categories are considered namely hot and cold zones.	Response Time	**Energy-efficient data-placement** through dividing servers into two major groups namely Hot and Cold zones. Energy can be saved through harnessing the idleness in the Cold zone.	Power models are used for the power levels, transitions times of power states and the subsystems access time including the disk, the processor and the DRAM.
Chen et al. [30]	MapReduce with interactive analysis (MIA) style workloads is classified using k-means clustering algorithm.	Response time	BEEMR (Berkeley Energy Efficient MapReduce) as an energy efficient MapReduce workload manager which is inspired by studying the Facebook Hadoop workload	Linear Power Model (CPU only).
Feller et al. [52]	Hadoop benchmarks including three micro-benchmarks namely TeraGen, TeraSort, and Wikipedia data processing are studied and the approach is applicable for both Bare metal and System level.	Application's completion time.	Achieved through considering the resource boundness of the tasks along with the differences between the map and reduce tasks.	Energy metered environment is utilized (Grid5000).
Lee et al. [106]	Bare Metal and System level Workflow applications Workload Workload Characteristics is assumed to be a prior knowledge	Makespan	Improved the resource utilization.	-

Table 2.5: Energy Efficient Research Considering Bare Metal Environment

Authors	Workload	SLA	Energy Saving	Energy Model
Pietri et al. [133]	Scientific Workflow applications	Makespan	**DVFS** on CPU component	Energy consumption p_f is estimated considering each processors operating frequency f through a cubic model derived in [132] $P_f = P_{base} + Pdif * (\frac{f - f_{base}}{f_{base}})^3$ and the total power consumption is estimated adding P_f to the power consumed when the processor is idle.
Durillo et al. [46]	Workflow Applications	Makespan	**Efficient resource allocation**	Energy consumption of task A running on multi core systems is estimated through: $c.E_{core} + E_{share}$. Here, c is the number of active cores and E_{core} is the energy consumed in active cores while executing task A. E_{shared} is the energy consumption of all the shared subsystems which are active during the task execution.

Table 2.6: Energy Efficient Research Considering OS-Level Virtualization

Authors	Workload	SLA	Energy Saving	Energy Model
Dong et al. [45]	Google Cluster Data (OS Container)	-	**DVFS** **Container Placement**	Proposed a power modeled named as VPC
Pandit et al. [126]	Synthetic workload (OS Container)	-	**Container placement**, Simulated Annealing is applied.	-
Hindman et al. [81]	Synthetic workload (OS Container)	Response Time	**Resource sharing between various programming models**	
Spicuglia et al. [147]	Primary Big Data application workloads (Shark [48], YARN [158]), and the background applications computing π. (Application Container)	Throughput	**Power Capping** (Considers a power constraint for running applications.)	Linear Power Model (CPU only)
Anselmi et.al [5]	Three Tier Application workload (Application Container)	Response Time	**Efficient allocation of resources**	-
Rolia et al. [138]	Enterprise application workload (Application Container)	-	**Efficient Allocation of Resources**, Servers with fast migration ability is used	-
Mohan et al. [116]	Request arrival for web servers that follows Poisson distribution (Application Container)	Allowable pending requests for each application in the dispatcher queue	-	Power Consumptoin of servers based on the number of VMS

Table 2.7: Energy Efficient Research Considering System-Level Virtualization

Authors	Workload	SLA	Energy Saving	Energy Model
Urgaonkar et al. [156]	Synthetic Workloads that follow a uniform random distribution is studied. The proposed approach is independent of workload prediction and its statistics as it uses the queueing information to learn and adapt to unpredictable changes in the workload pattern	Throughput	**DVFS** (on CPU component)	The power-frequency relationship is presented as a quadratic model $P(f) = P_{min} + \alpha(f - f_{min})^2$.
Gmach et al. [67]	Enterprise application workloads analysis is utilized for estimating the under-load and over-load situations for hosts	Response Time	**VM Consolidation** (When to migrate)	Linear Power Model (CPU only)
Beloglazov et al. [13]	PlanetLab Worklaod	The time during which host experience is identified overloaded	**VM Consolidation** (What triggers migration with fixed Under-load and Over-load thresholds)	Linear Power Model (CPU only)
Beloglazov et al. [14]	PlanetLab Workload	The time during which host experience is identified overloaded	**VM Consolidation** (What triggers migration with automatic under-load and overload detection algorithms along with two VM selection approaches)	Linear Power Model (CPU only)
Beloglazov et al. [15]	PlanetLab Workload, The approach is independent of workload including stationary and non-stationary workloads	A QoS goal as an input of consolidation algorithm	**VM Consolidation** (What triggers migration with optimally adjusting the overload threshold and selecting the right VM to migrate)	Linear Power Model (CPU only)
Meng et.al [113]	VM workloads of a commercial data center	Response time (Considering a performance constraint for each VM)	**VM Consolidation** (Where to migrate considering the Statistical Multiplexing approach)	-

Table 2.8: Energy Efficient Research Considering System-Level Virtualization

Authors	Workload Characterization	SLA	Energy Saving	Energy Model
Moreno et al. [119]	Google Workload is Classified based on task resource Usage patterns	QoS is insured through placing VMs based on their interference with the co-located VMs	**VM Consolidation** Where to migrate VMs? Interference Aware VM Placement	Linear Power Model (CPU only)
Cagar et al. [23]	Google Workload	Response Time	**VM Consolidation** Where to migrate VMs? Interference Aware VM Placement	-
Chen et al. [29]	The workload data of 5,415 servers from ten different companies is characterized via fitting distributions	A probabilistic function that contains the probability of a host being overloaded	**VM Consolidation** Where to migrate VMs? Statistical Multiplexing VM Placement	-
Verma et al. [159]	Enterprise application workload from the production data center of a multi-national Fortune Global 500 company. A detail analysis of the workload is presented including the correlation between the workloads.	Number of time instances that an application demand is more than the server's capacity	**VM Consolidation** Where to migrate VMs? Statistical Multiplexing VM Placement	The power models are derived from actual measurements on the servers (Power vs CPU utilization).
Calheiros et al. [24]	Bag of tasks applications considering CPU Intensive tasks	User defined deadlines	**DVFS** (CPU only)	The power consumption of each host is calculated considering the contribution of each CPU core.
Laszewski et al. [160]	Worldwide LHC Computing Grid (WLCG) workload	Throughput	**DVFS** (CPU only)	Energy is estimated considering CPU frequency

Table 2.9: Energy Efficient Research Considering System-Level Virtualization

Authors	Workload Characterization	SLA	Energy Saving	Energy Model
Kim et.al [97]	Analyzed the characteristics of scale-out applications obtained from a data center.	Virtual machines are provisioned according to their peak load.	**DVFS** **Efficient resource allocation** placing VMs based on correlation analysis	Energy Model in [128].
Tomas et al. [153]	Real-life interactive workloads and non-interactive batch applications	Response Time	**Overbooking resources**	-
Klein et al. [101]	RUBiS and RUBBoS	Response time and throughput	**Overbooking resources**	-
Forestiero et al. [57]	Logs of Eco4Cloud company `www.eco4cloud.com`	Avoiding any sites from being overloaded	**Consolidating workloads** on a multi-site platform (Geo-distributed cloud environment) Where to migrate?	-
Khosravi et al. [95]	Lublin-Feitelson workload model is employed to generate the Web application and Bag-of-Tasks workload.	-	**Efficient placement of VMs**	Modeled considering the frequency of CPU.
Assuncao et al. [8]	747 VM request workloads are classified.	Makes sure that most of the workloads can fit into selected templates	**Efficient allocation of resources** Decreasing the number of required templates	-

Table 2.10: Energy Efficient Research Considering Hybrid Virtual Environment

Authors	Workload	SLA	Energy Saving	Energy Model
Dhyani et al. [41]	Synthetic Workload	-	**Efficient allocation of resources**	-
Bilcher et al. [17]	Traces from 30 dedicated servers hosting different types of application services	-	**Efficient resource allocation**, Decreasing the number of required servers for hosting applications	-
Tchana et al. [151]	An enterprise Internet application benchmark (SPECjms2007 benchmark `http://www.spec.org/jms2007/`)	Service degradation threshold defined at start time for each application	**Software Consolidation**	-
Yaqub et al. [168]	High variability datasets from Google are characterized for OpenShift cloud which can span multi-domain IaaS.	SLA violation (SLAV) is modeled as upper bound estimate considering performance degradation due to both migration (PDM) and contention on machines resources (PDC)	**Software Consodilation**, Container migration	Linear Power Model (CPU only)
Almeida et al. [4]	Transactional web services workload is classified into independent Web service (WS) classes.	Response time	**Efficient resource allocation**	-

Chapter 3
Virtual Machine Customization and Task Mapping Architecture

Energy usage of large-scale data centers has become a major concern for cloud providers. There has been an active effort in techniques to minimize the energy consumed in data centers. However, most approaches lack the analysis and application of real cloud backend traces. The focus of existing approaches is on virtual machine migration and placement algorithms, with little regard to tailoring virtual machine configuration to workload characteristics, which can further reduce the energy consumption and resource wastage in a typical data center. To address these weaknesses and challenges, in this chapter we propose a new architecture for cloud resource allocation that maps groups of tasks to customized virtual machine types. This mapping is based on task usage patterns obtained from the analysis of historical data extracted from utilization traces. Further, the proposed architecture is extended to incorporate the recently introduced Container as a Service (CaaS) and the impact of workload study and the selected clustering feature set on the efficiency of our proposed VM sizing technique is investigated. When the right feature set is used for the workload study, the experimental results showed up to 7.55% and 68% improvements in the average energy consumption and total number of instantiated VMs respectively when compared to baseline scenarios where the virtual machine sizes are fixed.

This chapter is derived from:

1. **Sareh Fotuhi Piraghaj**, Rodrigo N.Calheiros, Jeffery Chan, Amir Vahid Dastjerdi , and Rajkumar Buyya "A Virtual Machine Customization and Task Mapping Architecture for Energy Efficient Allocation of Cloud Data Center Resources", *The Computer Journal, vol. 59, no. 2,Pages. 208-224, 2016.*
2. **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "Efficient Virtual Machine Sizing For Hosting Containers as a Service," *Proceeding of the 2015 IEEE World Congress on Services (SERVICES2015), Pages. 31 - 38, New York, United States.*

3.1 Introduction

As stated by Armbrust et al. [6], cloud computing has the potential to transform a large part of the IT industry while making software even more attractive as a service. However, the major concern in cloud data centers is the drastic growth in energy consumption, which is a result of the rise in cloud services adoption and popularity. This energy consumption results in increased Total Cost of Ownership (TCO) and consequently decreases the Return of Investment (ROI) of the cloud infrastructure.

There has been a growing effort in decreasing cloud data centers' energy consumption while meeting Service Level Agreements (SLA). Since servers are one of the main power consumers in a data center [174], in this chapter we mainly focus on the efficient utilization of computing resources.

Virtualization technology is one of the key features introduced in data centers that can decrease their energy consumption. This technology enables efficient utilization of resources and load balancing via migration and consolidation of workloads. Therefore, a considerable amount of energy is saved with virtual machine migrations from underloaded servers by putting them in a lower power state. Many approaches utilize this technology, along with various heuristics, concentrating solely on virtual machine migrations and VM placement techniques with the objective of decreasing the data center power consumption. However, these approaches ignore tailoring virtual machine configurations to workload characteristics and the effect of such tailoring on the energy consumption and resource wastage in a typical data center. User-defined virtual machine configuration is an available option for most cloud service models such as Google[1]. Therefore, one of the challenges is to propose a method for defining the most efficient virtual machine configuration for a given application.

Apart from VM configuration, the other factor impacting the efficiency of resource utilization is the application of the knowledge obtained from the analysis of the real world clouds trace logs. This analysis enables an understanding of the variance of workloads that should be incorporated in solutions, as they affect the performance of proposed resource management approaches.

[1]https://cloud.google.com/compute/docs/instances/creating-instance-with-custom-machine-type

In this chapter, we propose an end-to-end architecture for energy-efficient resource allocation and management in data centers. Because of the predefined software and applications that are executed in the data center, there exist similarities between the usage patterns of tasks and hence similar tasks can be grouped together using clustering algorithms. Our proposed solution decreases the resource wastage in data centers via virtualization and efficient resource allocation policies.

This chapter addresses the problem of inefficient resource allocation by tailoring VM configurations to the workload along with the determination of the maximum number of tasks that can be allocated to each VM type. For determination of virtual machine types and their capacity, we leverage similarities in the utilization patterns reported in the Google traces that is confirmed by previous studies [31, 43, 115, 146]. These similarities enable tasks to be grouped based on average resource usage via clustering techniques and consequently decrease the granularity of resource allocation problem. In this respect, decisions are made for each group of tasks instead of per task. Further, the output of clustering is used for determination of customized virtual machine types. The actual resource utilization of tasks is considered during the grouping process, since there is a considerable gap between the actual reported resource usage and the requested amount of resources for task execution in the studied trace. In this respect, considering the actual resource utilization during task execution will result in less resource wastage and consequently less energy consumption, which is one of the objectives of the proposed architecture. An evaluation of the proposed architecture shows that the policy that considers the actual reported usage results in less energy consumption in the data center. This policy exhibited 73% improvements when comparing to a policy that allocates the resources of virtual machines based on the resource estimation provided by users.

This chapter also studies the efficiency of determined VM sizes considering the data center power consumption. Moreover, it considers the Container as a Service (CaaS) cloud service model that is discussed in Chapter 2. This service model has been introduced in 2014 by Google and Amazon in addition to the traditional cloud services, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). An example of container management system is docker[2], which is a tool

[2]Docker:https://www.docker.com/

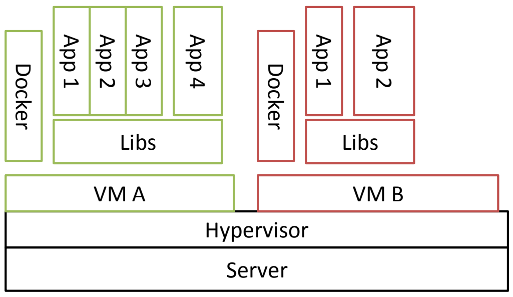


Figure 3.1: A Simple CaaS Deployment Model on IaaS.

that allows developers to define containers for applications.

Like traditional service models, to reduce the energy consumption of CaaS one may choose virtual machine consolidation, Dynamic Voltage and Frequency Scaling (DVFS), or both of them combined. However, as we discussed these efforts would be in vain if VM sizes are not customized to better support deployed containers. Figure 3.1 illustrates a situation where the size of *VM B* is not optimally allocated to the containers when compared to *VM A* . As a result, there is resource wastage that results in inefficiency in terms of energy consumption regardless of how effective and energy efficient is the VM consolidation technique in place. In other words, as we discussed in Chapter 2, VM consolidation is limited by other resources such as memory (Figure 2.5), hence for improving the resource utilization of the servers, resources of virtual machines should also be allocated and utilized efficiently.

In this regard, we extended our proposed architecture to incorporate the CaaS cloud model. In this methodology, we added one extra step between task grouping and VM type identification. In this extra step, the clustering output is used for mapping tasks to containers. Each task is assumed to represent a container with the same resource requirement as the task itself. Afterward, each cluster of containers is mapped to a corresponding virtual machine type.

Using our extended architecture, we investigate the impact of feature set selection on the number of resulting clusters along with the effect on resource allocation efficiency in a CaaS cloud service model. We also compare our VM sizing technique with fixed VM size baseline scenarios. The experimental results show that selecting the right features improves the efficiency of clustering process and consequently results in more energy savings. In summary, our proposed approach results in less number of servers, which in turn results in less energy consumption in the data center.

In order to apply information of real cloud backend traces in our solutions and in their evaluation, we utilized Google traces. The first Google log provides the normalized resource usage of a set of tasks over a 7-hour period. The second version of the Google traces, which was released in 2012, contains more details in a longer time frame. Therefore, the data set used in this chapter is derived from the second version of the Google cloud trace log [137] collected during a period of 29 days. The log consists of data tables describing the machines, jobs, and tasks.

Recent work analyzing Google traces focused on various objectives such as characterization of task usage [171], task grouping for workload prediction and capacity planning [115], characterization of applications [43], modeling and synthesis of task placement constraints [142], and workload characterization for simulation parameter extraction and modeling [42, 118, 146]. Our work in this chapter contributes to the current research area by introducing an architecture that utilizes the knowledge obtained from the characterization of task usage patterns to determine efficient resource allocation. In summary, the key contributions of this chapter are:

1. We propose an end-to-end architecture for efficient allocation of requests on data centers that reduces the infrastructure's energy consumption.

2. We present an approach, applied to the proposed architecture, to identify virtual machine configurations (types) in terms of CPU, memory, and disk capacity via clustering tasks, taking into consideration usage patterns of each cluster. The aforementioned architecture is then extended to incorporate the CaaS service model.

3. We propose an approach for identification of VM task capacity, which is the maximum number of tasks that can be accommodated in a virtual machine, considering different estimates, including the average resource usage of tasks in each cluster.

4. We investigate the impact of feature set selection on the number of resulting clusters along with the effect on the resource allocation efficiency.

5. We compare our VM sizing technique with fixed VM size baseline scenarios.

3.2 Related Work

There is a vast body of literature that considers power management in virtualized and non-virtualized data centers via hardware and software-based solutions [89,98,123]. Most of the works in this area focus on host level optimization techniques neglecting energy-efficient virtual machine size selection and utilization. These approaches are suitable for IaaS cloud services, where the provider does not have any knowledge about the applications running in every virtual machine. However, for SaaS and CaaS service models, information about the workload on the virtual machines and improvements on the size selection and utilization efficiency on VM level could be the first step towards more energy efficient data centers, as the results presented in this chapter demonstrate.

Regarding the comparison between hosting SaaS on bare-metal servers or virtual machines, Daniel et al. [66] explored the differences between fixed virtual machine sizes and time shares. Although they concluded that the time-share model requires less number of servers, they have not considered dynamic VM size selection in their experiments. Similarly, in the SEATS (smart energy-aware task scheduling) framework, Hosseinimotlagh et al. [83] introduced an optimal utilization level of a host to execute tasks that minimizes the energy consumption of the data center. In addition, they also presented a virtual machine scheduling algorithm for maintaining the host optimal utilization level while meeting the given QoS.

Apart from the level of optimization (host-level, data center-level, or virtualization level), most of the research in the area lack the analysis of real cloud backend traces and the variance in the cloud workload in the proposed solutions. In 2009, Yahoo! released traces from a production MapReduce cluster to a selection of universities. In the same year, Google made the first version of its traces publicly available. Google trace's release resulted in a number of research investigating the problems of capacity planning and scheduling via workload characterization and statistical analysis of the planet's largest cloud backend traces [137]. Hence, we utilized the second version of Google traces to validate our proposed architecture and our experiments.

Different from previous works, we leverage virtualization and containerization technology together and map the groups of tasks to containers and containers to VMs. The configuration of the container-optimized VMs is chosen based on the workload characteristics, which results in less resource wastage. In our methodology, the problem of high energy consumption that results from low resource utilization is also addressed, which is not explored in most of the previous studies. Further, we detail the research works performed on Google cluster data.

3.2.1 Google Trace Research Works

Next, we discuss in more details related research that studied or applied Google trace data. The works in this area fall into three major categories, namely statistical analysis, workload modeling and characterization, and simulation and modeling.

Statistical Analysis

The first version of the Google traces contains the resource consumption of tasks, whereas the second version of Google traces covers more details including machine properties and task placement constraints. These constraints limit the machines onto which tasks can be scheduled [137]. In order to measure the performance impact of task placement constraints, Sharma et al. [142] synthesized these constraints and machine properties into performance benchmarks of Google clusters in their approaches.

Garraghan et al. [64] investigated server characteristics and resource utilization in the Google cluster data. They also explored the amount of resource wastage resulted from failed, killed, and evicted tasks for each architecture type over different time periods. The average resource utilization per day lies between 40-60% as stated by Reiss et al. [136], and the CPU wastage on average server architecture type lies between 4.52-14.22%. These findings justify an investigation of new approaches for improving resource utilization and reducing resource wastage.

Di et al. [44] investigated the differences between a cloud data center and other Grids and cluster systems considering both workload and host load in the Google data center. An analysis of the job length and jobs resource utilization in various system types, along with job submission frequency, shows that the host load in a cloud environment faces higher variance resulted from higher job submission rate and shorter job length. As a result, the authors identified three main differences between cloud and Grid workloads:

firstly, Grid tasks are more CPU intensive, whereas cloud tasks consume other resources, such as memory, more intensively. Secondly, CPU load is much noisier in clouds than in Grids. Thirdly, the host load stability differs between infrastructures, being less stable in clouds. These differences make the analysis of cloud traces crucial for researchers, enabling them to verify the applicability of heuristics in real cloud backend environments.

Workload Modeling and Characterization

Mishra et al. [115] and Chen et al. [31] explored the first version of the Google cluster traces and two approaches were introduced for workload modeling and characterization. Mishra et al. [115] used the clustering algorithm K-means for forming groups of tasks with more similarities in resource consumption and duration. Likewise, Chen et al. [31] used K-means as the clustering algorithm. In their experiments, the authors classified jobs[3] instead of tasks. Di et al. [43] characterized applications, rather than tasks, running in the Google cluster. Similarly to the two previous approaches, the authors chose K-means for clustering, although they optimized the K-means result using the Forgy method.

Moreno et al. [118] presented an approach for characterization of the Google workload based on users and task usage patterns. They considered the second version of the Google traces and modeled the workload for two days of it. Later in 2014 [146], authors extended the work with an analysis of the entire tracelog. The main contribution of the work is considering information about users along with the task usage patterns. Moreno et al. [118, 146] also used K-means for grouping purposes. They estimated the optimal k with the quantitative approach proposed by Pham et al. [131].

The previous study demonstrated that there are similarities in task usage patterns of Google backend traces. Therefore in the architecture we introduce later in this chapter, likewise previous approaches [31, 115], tasks with similarities in their usage patterns are grouped using clustering. In typical clustering, the number of clusters is a variable that is data-dependent and has to be set beforehand. Approaches in [118, 146] use K-means and vary the number of clusters considering a finite range for example 1 to 10. Then, the optimal value of k is derived considering the degree of variability in derived clus-

[3] A job is comprised of one or more tasks [137].

ters [118,146] and Within cluster Sum of Squares (WSS) [43]. Although these approaches could be applied here, we aimed to make the architecture as autonomous as possible and thus we avoided manual tuning of the number of clusters for each dataset like previous studies [43,118,146]. Pelleg and Moore [129] proposed X-means, a method that combines K-means with BIC. The latter is used as a criterion for automatic selection of the best number of clusters. Hence, we utilize X-means rather than existing approaches based solely on K-means [43,118,146]. It is worth mentioning that the workload modeling part of the architecture can be substituted, without changes in other components of the proposed architecture, by other approaches available in the literature [31,43,115,118,146].

The concept of task clustering has been previously investigated and shown to be effective outside of cloud computing area [120,145,162]. Our approach is different from them in terms of the objective and the target virtualized environment. For example, Singh et al. [145] and Muthuvelu et al. [120] utilized the technique for reducing communication overhead for submission of tasks in Grid systems, which are geographically distributed, in contrast with our application for energy minimization in a centralized cloud data center. Task clustering is also utilized by Wang et al. [162] to improve energy efficiency in clusters via dynamic frequency and voltage (DVFS) techniques targeting parallel applications. Our approach, on the other hand, is agnostic to the application model and achieves energy-efficiency via consolidation and efficient utilization of data center resources. Furthermore, our work goes beyond these previous approaches on clusters and Grids by leveraging virtualization and considering the newly introduced CaaS cloud service model.

Simulation and Modeling

Di et al. [42] proposed GloudSim as a distributed cloud simulator based on Google traces. This simulator leveraged virtualization technology and modeled jobs and their usage in terms of the CPU, memory, and disk. It supports simulation of a cloud environment that is as similar as possible to Google cluster.

Moreno et al. [118, 146] proposed a methodology to simulate the Google data center. Authors leveraged their modeling methodology to build a workload generator. This generator is implemented as an extension of the well-known cloud discrete simulator

CloudSim [26] and is capable of emulating the user behavior along with the patterns of requested and utilized resources of submitted tasks in Google cloud data center.

In this chapter, we present an end-to-end architecture aiming at efficient resource allocation and energy consumption in cloud data centers. In this architecture, the cloud provider utilizes the knowledge obtained from the analysis of the cloud backend workload to define customized virtual machine configuration along with maximum task capacity of virtual machines.

In the proposed architecture, likewise the discussed related work [42, 118, 146], we assume the availability of virtualization technology and CaaS cloud service model where tasks are executed on top of containers while containers are running inside virtual machines instead of physical servers. This architecture can also be implemented utilizing the aforementioned simulation models [42, 118, 146]. Our work is different since we aim at decreasing energy by defining the virtual machines configurations along with their maximum task capacity.

3.3 System Model and Architecture

Our proposed architecture targets Platform as a Service (PaaS) data centers operating as a private cloud for an organization. Such a cloud offers a platform where users can submit their applications in one or more programming models supported by the provider. The platform could support, for example, MapReduce or Bag of Tasks (BoT) applications. Here, users interact with the system by submitting requests for execution of applications supported by the platform. Every application, in turn, translates to a set of jobs to be executed on the infrastructure. In our studied scenario, the job itself can be composed of one or more tasks.

3.3.1 User Request Model

In the proposed model, users of the service submit their application along with estimated resources required to execute it and receive back the results of the computation. The exact infrastructure where the application executes is abstracted away from users. Parameters of a task submitted by a user are:

- Scheduling Class;
- Task priority;
- Required number of cores per task;
- Required amount of RAM per task; and
- Required amount of storage per task.

All the aforementioned parameters are present in Google Cluster traces [137].

3.3.2 Cloud Model

In the presented cloud model, system virtualization technology [10] is taken into consideration. This technology improves the utilization of resources of physical servers by sharing them among virtual machines [170]. Apart from this, live migration of VMs and overbooking of resources via consolidation of multiple virtual machines in a single host reduce energy consumption in the data center [16]. The other benefit of virtualization is the automation it provides for application development [32]. For example, once a virtual machine is customized for a specific development environment, the VM's image can be used on different infrastructures without any installation hassles. Therefore, as long as the virtual machine is able to be placed on the server, homogeneity of the environment offered by the VM image is independent of the physical server and its configuration. These characteristics and advantages of the virtualization technology persuaded us in applying this in our proposed architecture.

Our focus is on data centers that receive task submissions and where tasks are executed in virtual machines instead of physical servers, a model that has been widely explored in the area of cloud computing [50, 157]. Since these tasks might be different in terms of running environments, it is assumed that tasks run in containers [137] that provide these requirements for every one of them. However, in our model, these containers run inside the virtual machines instead of the physical machines. This can be achieved with the use of Linux containers or tools such as Docker [114], an open platform for application development in which containers can run inside the virtual machine or on physical hosts.

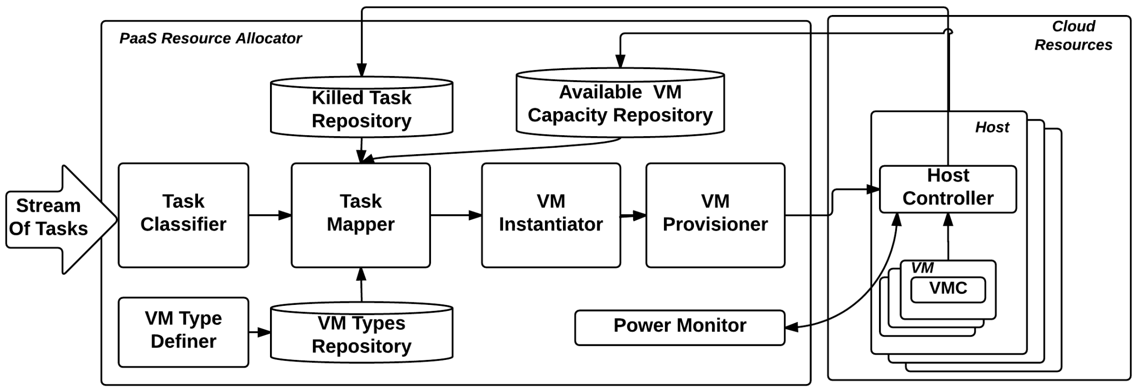


Figure 3.2: Proposed system architecture and its components.

3.3.3 System Architecture

The objective of the proposed architecture (shown in Figure 3.2) is to execute the workload with minimum wastage of energy. Therefore, one of the challenges is finding optimal VM configurations, in such a way that the accommodated tasks have enough resources to be executed and resources are not wasted during the operation. Since the proposed model has been designed to operate in a private cloud, the different number and types of applications can be controlled and there is enough information about submitted tasks so that cloud usage can be profiled.

3.3.4 System Components

The proposed architecture is presented in Figure 3.2 and their components are discussed in the rest of this section.

Pre-execution Phase

We discuss the components of the proposed architecture that need to be tuned or defined before system runtime.

- **Task Classifier:** This component is the entry point of the streaming of tasks being processed by the architecture. It categorizes tasks arrived in a specified time frame into predefined classes. The classifier is trained with the clustering result of the historical data before system startup. The clustering is performed considering average

CPU, memory, and disk usage together with the priority, length, and submission rate of tasks obtained from the historical data. The time interval for the classification process is specified by the cloud provider according to the workload variance and task submission rate. Once an arriving task is classified in terms of the most suitable virtual machine type for processing it, the task is forwarded to the Task Mapper to proceed with the scheduling process. The Task Mapper component is discussed in the execution phase.

- **VM Type Definer:** This component is responsible for defining the configurations of virtual machines based on the provided historical data. Determining the optimal VM configuration requires analysis of task usage patterns. In this respect, the identification of groups of tasks with similar usage patterns reduces the complexity of estimating the average usage for new tasks. These patterns, which identify groups of tasks that have a mutual optimal VM configuration, are obtained with application of clustering algorithms.
- **VM Types Repository:** In this repository, the available virtual machine types, including CPU, memory, and disk characteristics, are saved. These types are specified by the *VM Type Definer* considering workload specifications and is derived from historical data used for training the task classifier component.

Execution Phase

The components that operate during the execution phase of the system are discussed below.

- **Task Mapper:** The clustering results from the *Task Classifier* are sent to the *Task Mapper*. The Task Mapper operation is presented in Algorithm 1. Based on available resources in the running virtual machines and the available VM types in the *VM Types Repository*, this component calculates the number and type of new virtual machines to be instantiated to support the newly arrived tasks. Apart from new VM instantiation when available VMs cannot support the arriving load, this component also reschedules rejected tasks that are stored in the killed task repository to the available virtual machines of the type required by the VM (if any). This

Algorithm 1: Overview of the Task Mapper operation process.

```
Input: KilledTasks, AvailablevmCapacity, NewTasks, VMTypeRepository
Output: NumberofvmsToInstatiate
1  foreach ProcessingWindow do
2      foreach Task in NewlyArrivedTasks do
3          if There is a vm in AvailablevmCapacity then
4              vm.Assign(Task)
5              vm.CheckStatus
6              Delete Task from NewlyArrivedTasks
7      foreach Task in KilledTasks do
8          if There is a vm in AvailablevmCapacity then
9              vm.Assign(Task)
10             vm.CheckStatus
11             Delete Task from KilledTasks
12     LeftTasks = Append KilledTasks to NewlyArrivedTasks
13     foreach Task in LeftTasks do
14         Calculate the NumberofvmsToInstantiate
```

component prioritizes the assignment of newly arrived tasks to available resources before instantiating a new virtual machine. However, in order to avoid starvation of the rejected tasks, the component assigns the newly arrived tasks to the available virtual machines and the killed tasks are assigned to newly instantiated VMs. The operation of this component on each processing window (Algorithm 1) yields complexity $O(n \times m)$, where n is the total number of tasks to be mapped (i.e., tasks in the *KilledTaskDictionary* along with the tasks received in the processing window) and m is the number of VMs.

- **Virtual Machine Instantiator:** This component is responsible for the instantiation of a group of VMs with the specifications received from the *Task Mapper*. This component decreases the start-up time of the virtual machines by instantiating a group of VMs at a time instead of one VM per time.

- **Virtual Machine Provisioner:** This component is responsible for determining the placement of each virtual machine on available hosts and turning on new hosts if required to support new VMs.

- **Killed Task Repository:** Tasks that are rejected by the Controller are submitted to

Algorithm 2: Virtual Machine Controller Process.

```
Input: $RunningTaskList$, $TaskUsage$
Output: $CPUUsage$,$MemoryUsage$,$DiskUsage$, $KilledTasksList$
1 foreach $Processingwindow$ do
2     foreach $Task$ in $RunningTaskkList$ do
3         $vm$.updateUsage()
4         $vm$.updateState()
5     foreach $vm$ whose state is $OverLoaded$ do
6         foreach $Task$ in $RunningTaskkList$ do
7             if $TaskPriority$ equals to $LowestPriority$ and has $MinNumberofKills$ then
8                 $vm$.killTask()
9                 $vm$.updateState()
```

this repository, where they stay until the next upcoming processing window to be rescheduled by the *Task Mapper*.

- **Available VM Capacity Repository:** IDs of virtual machines that have available resources are registered in this repository. It is used for assigning tasks killed by the *Virtual Machine Controller*, along with newly arrived ones, to available resource capacity.

- **Power Monitor:** This component is responsible for estimating the power consumption of the cloud data center based on the resource utilization of the available hosts.

- **Host Controller:** It runs on each host of the data center. It periodically checks virtual machine resource usage (which is received from the Virtual Machine Controllers) and identifies underutilized machines, which are registered in the available resource repository. This component also submits killed tasks from VMs running on its host to the *Killed Task Repository* so that these tasks can be rescheduled in the next processing window. This component also sends the host usage data to the *Power Monitor*.

- **Virtual Machine Controller (VMC):** The VMC runs on each VM of the cloud data center. It monitors the usage of the VM and if the resource usage exceeds the virtual machine capacity, it kills a number of tasks with low priorities so that high priority ones can obtain the resources they require in the virtual machine. In order to avoid task starvation, this component also considers the number of times a task has been killed. The Controller sends killed tasks to the *Host Controller* to be submitted to the global killed task repository. As mentioned before, killed tasks are then rescheduled on an available virtual machine in the next processing window. The operation of this component is shown in Algorithm 2 that has a time complexity of $O(n \times m)$, where n is the number of running tasks and m is the number of VMs.

3.4 Task Clustering

In this section, we discuss the selected clustering feature set and the clustering algorithm utilized for clustering tasks with more details.

3.4.1 Clustering Feature Set

As our feature set, we used the following characteristics of each task:

- **Task Length**: The time during which the task was running on a machine.
- **Submission Rate**: The number of times that a task is submitted to the data center.
- **Scheduling Class**: This feature shows how latency sensitive the task/job is. In the studied traces, the scheduling class is presented by an integer number between 0 and 3. Tasks with a 0 scheduling class are non-production tasks. The higher the scheduling class is, the more latency sensitive is the task.
- **Priority**: The priority of a task shows how important a task is. High priority tasks have preference for resources over low priority ones [137]. The priority is an integer number between 0 and 10.
- **Resource Usage**: The average resource utilization U_T of a task T in terms of CPU, memory, and disk, which is obtained using Equation 3.1. In this equation, nr is the number of times that the task usage (u_T) is reported in the studied 24 hours period and $u_{(T,m)}$ is the *m-th* observation of the value of utilization u_T in the traces.

$$U_T = \frac{\sum_{m=1}^{nr} u_{(T,m)}}{nr} \tag{3.1}$$

The selected features of the data set were used for estimation of the number of task clusters and determination of the suitable virtual machine configuration for each group.

3.4.2 Clustering Algorithm

Clustering is the process of grouping objects with the objective of finding the subsets with the most similarities in terms of the selected features. In this respect, both the objective of the grouping and the number of groups affect the results of clustering. In our specific approach, we focus on finding groups of tasks with similarities in their usage pattern so that available resources can be allocated efficiently. For determining the other attribute, the namely definition of the most effective number of clusters, the X-means algorithm is utilized.

X-means Clustering Algorithm

Pelleg et al. [129] proposed the X-means clustering method as the extended version of K-means [76]. In addition to grouping, X-means also estimates the number of groups present in a typical dataset, which in the context of the architecture is the incoming tasks. K-means is a computationally efficient partitioning algorithm for grouping n-dimensional dataset into k clusters by minimizing within-class variance. However, supplying the number of groups (k) as an input of the algorithm is challenging since the number of existing groups in the dataset is generally unknown. Furthermore, as our proposed architecture aims for automated decision making, it is important that the number of input parameters is reduced and that the value of k is automatically calculated by the platform. For this reason, we opted for X-means.

As stated by Pelleg et al. [129], X-means efficiently searches the space of cluster locations and number of clusters in order to optimize the Bayesian Information Criterion (BIC). BIC is a criterion for selecting the best fitting model amongst a set of available models for the data [141]. Optimization of the BIC criterion results in a better fitting model.

X-means runs K-means for multiple rounds and then clustering validation is performed using BIC to determine the best value of k. It is worth mentioning that X-means

Algorithm 3: Estimation of the optimum number of tasks for a VM Type.

```
input: $ClusterofTasks$,
       $nT$: Maximum number of tasks per VM,
       $nI$: Number of iterations
Output: $NumberofTasksPerCluster$
1  foreach $ClusterofTasks$ do
2      $AvgCPU \leftarrow$Average CPU Usage of the $ClusterofTasks$
3      for $k$ from 1 to $nI$ do
4          for $i$ from 1 to $nT$ do
5              $ClusterSample \leftarrow i$ random samples of $TaskCluster$ without replacement
6              $AvgCPU_s \leftarrow$ Average CPU usage for the $ClusterSample$
7              $CPU_{Error} \leftarrow \frac{AvgCPU - AvgCPU_s}{AvgCPU}$
8              $temp_{error}[i] \leftarrow CPU_{Error}$
9          $Min_{Error}[k] \leftarrow$ Index of $min(temp_{Error})$
10     $NumberofTasksPerCluster \leftarrow mode(Min_{Error})$
```

has been successfully applied in different scenarios [47,73,91,144].

3.5 Identification of VM Types for the VM Type Repository

Once clusters that represent groups of tasks with similar characteristics in terms of the selected features are defined, the next step is to assign a VM type that can efficiently execute tasks that belong to the cluster. By efficiently, we mean successfully executing the tasks with minimum resource wastage. Parameters of interest of a VM are number of cores, amount of memory, and amount of storage. Since tasks in the studied trace need small amount of storage, the allocated disk for virtual machines are assumed to be 10 GB, which is enough for the Operating System (OS) installed on the virtual machine and the tasks disk usage.

3.5.1 Determination of Number of Tasks for each VM Type

Algorithm 3 details the steps taken for estimation of the number of tasks for each virtual machine type. In order to avoid overloading the virtual machines, the maximum number of tasks in each VM (nT) is set to 150. This amount is allowed to increase if the resource demand is small compared to the VM capacity. Then, for each allowed number of tasks *i*

(i between 1 and nT), i random tasks are selected from the cluster of task and the average CPU utilization is calculated for this selection. The CPU error is then reported and stored in $temp_{error}$.

Next, according to the $temp_{error}$, the algorithm finds the value of i that has the lowest CPU usage estimation error as the VM's number of tasks. This process is repeated for 500 (nI) iterations, which enables enough data to be collected for drawing conclusions. The VM's number of tasks in each iteration is then saved in Min_{error}. According to Min_{error}, the number of task for each VM type would be the number that shows the minimum estimation error in most of the iterations. In other words, the algorithm selects the number of tasks that is the most probable to result in less estimation errors.

3.5.2 Estimation of Resource Usage of Tasks in a Cluster

After estimating the maximum number of tasks in each virtual machine with the objective of decreasing the estimation error, the virtual machine types need to be defined. For this purpose, there is a need to estimate the resource usage of a typical task running in a virtual machine. For estimating the resource usage of each task in a cluster, the algorithm uses the average resource usage and variance of each cluster of tasks in our selected dataset. The first step for this is the computation of the average resource usage of each task during the second day of the trace. Then for each cluster, 98% confidence interval of the average utilization of resources of the tasks in the group is used. The upper-bound of the calculated confidence interval is then used as the estimate of the resource demands (RDs) for a typical task from a specific cluster.

3.5.3 Determination of Virtual Machines Configuration

After obtaining the estimates for resource demands (RD) and the number of tasks in a virtual machine type (nT), the specifications of the virtual machine is derived using Equation 3.2.

$$Capacity = \lceil nT * RD \rceil \tag{3.2}$$

Table 3.1: Virtual machine configurations.

VM Type	Number of Tasks	vCPU	Memory (GB)	VM Type	Number of Tasks	vCPU	Memory (GB)
TYPE 1	136	3	4.5	TYPE 10	250	1	0.4
TYPE 2	125	1	0.5	TYPE 11	188	3	1.6
TYPE 3	500	1	1.8	TYPE 12	1250	1	1.1
TYPE 4	38	6	11	TYPE 13	118	4	10.3
TYPE 5	139	5	3.4	TYPE 14	126	25	14.2
TYPE 6	250	1	0.9	TYPE 15	100	2	1.9
TYPE 7	143	14	20.6	TYPE 16	136	3	6.8
TYPE 8	150	3	2.4	TYPE 17	143	2	1.1
TYPE 9	154	8	4.3	TYPE 18	500	1	3.8

Since tasks running in one virtual machine are already sharing the resources, at least one core of the CPU of the physical machine is assigned to each virtual machine. Because of the rounding process in Equation 3.2, the number of tasks in each virtual machine is estimated again applying the same equation.

The above process was applied to determine VM types for each cluster. VM types resulting from the above process are stored in the VM Types Repository to be used by the Task Mapper for assignment purposes. The application of this process resulted in the VM types described in Table 3.1. The number of tasks nT obtained from Equation 3.2 is used as the virtual machines' task capacity for the proposed Utilization based Resource Allocation (URA) policy, which is briefly discussed in the next section along with the other proposed policies.

3.6 Resource Allocation Policies

The number of tasks residing in one VM varies from one cluster to another. As discussed in the previous section, virtual machine configurations are tailored to the usage pattern of the tasks residing in the VMs. The same virtual machine configurations are used for all the proposed policies. However, these algorithms are different in terms of the task capacity of the virtual machines for each cluster of tasks. These resource allocation policies are detailed below.

- **Utilization Based Resource Allocation** (URA): In this policy, the number of tasks assigned to each VM is computed according to the 98% confidence interval of the observed average utilization of resources by the tasks being mapped to the VM. For example, if historical data shows that tasks of a cluster used on average 1GB, and

Algorithm 4: Determination of the minimum number of running tasks for each virtual machine that causes VM resource utilization to be higher than 90% of its capacity without causing rejections.

```
Input: $vmListsofClusters = \{vmList_1, ..., vmList_{18}\}$
       $vmList_{clusterIndex} = \{vmID_1, ..., vmID_{numberofVMs}\}_{clusterIndex}$
       $resourceList = \{CPU, memory, disk\}$
Output: $nT_{(clustrIndex,Res)} = \{nt_{vmID_1}, ..., nt_{vmID_{numberofVMs}}\}$
1 for $clusterIndex \leftarrow 1$ to 18 do
2     $vmIDList \leftarrow vmListsofClusters.get(clusterIndex)$
3     for $vmID$ in $vmIDList$ do
4         foreach $Res$ in $resourceList$ do
5             Find minimum number of running tasks ($nt$) that caused the utilization
              of the resource ($Res$) to be between 90% to 100% of its capacity.
              $nT_{(clustrIndex,Res)}.add(nt_{vmID})$
```

tasks of such cluster are going to be assigned to a VM with 4 GB of RAM, URA will assign 4 of such tasks, regardless the estimated amount of memory declared by the user when submitting the corresponding job (which is the value obtained from the traces). The task capacity of each virtual machine type is equal to the nT term obtained from Equation 3.2.

- **Requested Resource Allocation** (RRA): In this policy, the same virtual machine types from URA are considered, however the number of tasks assigned to a VM is based on the average requested amount by the submitted tasks. As mentioned before, the requested amount of resources is submitted along with the tasks. RRA is used as a baseline for our further comparisons in terms of data center power consumption and server utilization.

The other four policies are derived from the results of the evaluation of URA. In this respect, the usage of virtual machines is studied to get more insights about the cause of rejections (CPU, memory, or disk) and the number of running tasks in each virtual machine when the rejections occurred.

For each virtual machine, the minimum number of running tasks that utilizes more than 90% of the VM's capacity in terms of CPU, memory, and disk without causing any rejections are extracted. This 90% limit avoids the occurrence of underutilized virtual machines. The extracted number is defined as $nt_{(vmID,resource)}$. The procedure is applied on each cluster and is explained with more details in Algorithm 4.

For each cluster determined by its $clusterIndex$ in Algorithm 4, nt is obtained for each VM type. Then, nt of the VMs in each cluster are gathered in a set named $nT_{clusterIndex,Res}$ for each of the considered resources (Res) including CPU, memory, and disk. We propose four policies to determine the number of tasks residing in each virtual machine. These policies as described below are based on the estimates (average, median, the first and the third quantile) derived from $nT_{clusterIndex,Res}$ for each cluster.

- **Average Resource Allocation policy (*AvgRA*):** For each cluster of tasks, considering m as the length of the set $nT_{clusterIndex,(CPU,memory,disk)}$, for the average number of tasks, we have:

$$nT_{Avg,resource} = (\sum_{i=1}^{m} nt_{i,resource}/_{m}) \tag{3.3}$$

  The nT_{Avg} is estimated for each resource separately. In this policy, the number of tasks residing in each virtual machine type is equal to the minimum nT obtained for each resource (Equation 3.4).

$$nT_{\text{minimum}} = \min(nT_{\text{Avg,CPU}}, nT_{\text{Avg,memory}}, nT_{\text{Avg,Disk}}) \tag{3.4}$$

- **First Quantile Resource Allocation policy (*FqRA*):** For this policy, the first quantiles[4] of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. Like AvgRA, the minimum amount obtained for each of the resources is used. By resource, we mean the virtual machine's CPU, memory, or disk capacity.

- **Median Resource Allocation policy (*MeRA*):** For this policy, the second quantiles (median) of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. Like the previous policy, the minimum amount of $nT_{Med,resource}$ obtained for each of the resources is used for determining the VM's task capacity.

- **Third Quantile Resource Allocation policy (*ThqRA*):** In this policy, the third quan-

[4]The k quantile of a sorted set is the value that cuts off the first $(25 * k)\%$ of the data. For first, second, and third quantile k is equal to 1, 2 and 3 respectively.

Table 3.2: Google Trace Data Tables [137]

Table Name	Description
Machine Events	Contains the machines specifications including normalized CPU, memory capacity, events , and platform ID
Machine Attributes	Describes the key-value pairs that indicates properties of the machines such as kernel version,clock speed and etc.
Job Events	Contains jobs and the lifecycles of jobs that is depicted in Figure 3.3
Task Events	Contains the tasks along with the lifecycles of tasks as depicted in Figure 3.3
Task Constraints	Contains the tasks placement constraints that restrict the type of machines that a task can be executed on.
Task Usage	Describes the actual normalized resource usage of tasks including the CPU, memory , and disk usage.

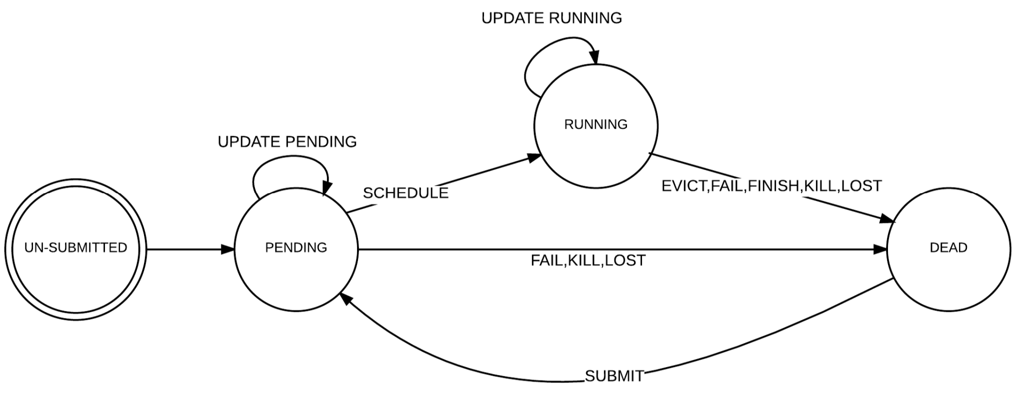


Figure 3.3: State Transition for jobs and tasks (Adopted from [137])

tiles of the $nT_{clusterIndex,Res}$ sets are used for determining the number of tasks allocated to each virtual machine type. As in the previous cases, the minimum amount of $nT_{Med,resource}$ obtained for each of the resources is used for determining the virtual machines task capacity.

3.7 Google Cluster Workload Overview

The dataset used in this chapter is derived from the second version of the Google cloud trace log [137] collected during a period of 29 days. The Google cluster log consists of data tables describing machines, jobs, and tasks as shown in Table 3.2. In the trace log, each job consists of a number of tasks with specific constraints. Considering these constraints, the scheduler determines the placement of tasks on the appropriate machines. The event type value in the job and tasks are reported in the event table. The event type value in the job and task event table shows the state of the job/task in the event (Figure 3.3). The job/task event has two types: events that change the scheduling state such as submitted, scheduled or running, and events that indicate the state of a job such as dead [137]. For the purpose of this evaluation, we utilize all the events from the trace log and we assume that all the events are occurring as reported in the trace log. The second day of the traces is selected for evaluation purposes, as it had the highest number of task submissions. Also, in order to eliminate placement constraint for tasks, we have chosen only one of the three platforms, the one with the largest number of task submissions.

In order to find the groups of tasks with same usage patterns, we utilized the reported data in the *Task Usage Table*. In this table, the actual task resource usage including normalized CPU, memory, and disk usage are reported for periods of five minutes. This data is used for clustering of tasks in order to find groups having the same usage pattern. The resource utilization measurements and requests are normalized, and the normalization is performed separately for each column.

We compare the cumulative distribution function (CDF) of requested and actual utilized resources including the CPU, memory, and disk in Figure 3.4 for the selected part of the Google workload. The average requested and utilized resources are calculated considering the tasks' requested and utilized reported resources during the second day of the Google traces. As it is depicted in Figure 3.4a, there is a considerable gap between the amount of requested and the utilized CPU. 45% of the tasks are requesting 6% of the biggest available CPU while the rest are requesting more than this amount. Looking at the actual CPU utilized by the submitted tasks during the studied period, 80% of the tasks are utilizing less than 1% of the biggest CPU, whereas the remaining tasks are utilizing almost 5% of the biggest available CPU.

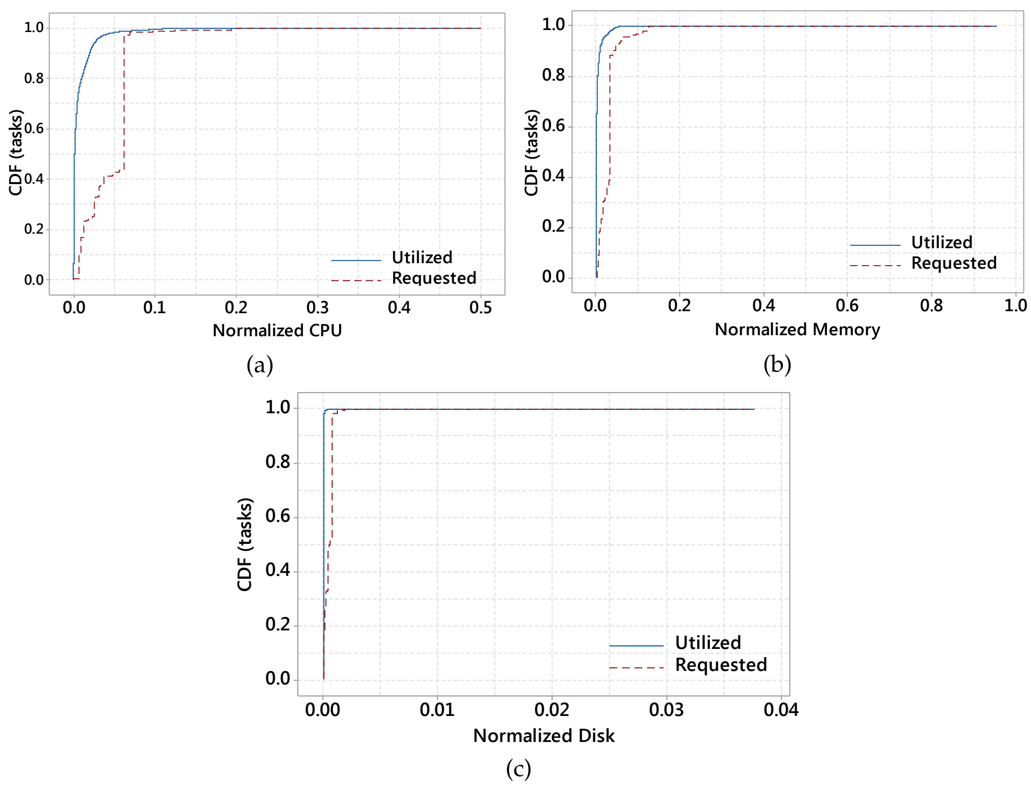


Figure 3.4: CDF of average requested and utilized resources for Google cluster tasks.

As depicted in Figure 3.4b, 80% of the tasks utilize less than 0.4% of the biggest available memory while they request more than 1% of the available memory. The same pattern can be observed for disk, as shown in Figure 3.4c, where tasks request for more than 0.07% of the biggest available disk while using a negligible amount of their requested disks.

In table 3.3, we detail the statistics of the studied parameters of the workload derived from the second day of the traces. These figures confirm our previous inferences regard-

Table 3.3: Workload Parameters and statistics during the 24 hours studied period.

Workload Parameters	Mean	StDev	Minimum	Maximum
Requested CPU	0.045189	0.027699	0	0.5
Utilized CPU	0.006637	0.013751	0	0.261725
Requested Memory	0.029162	0.022123	0.000001	0.9551
Utilized Memory	0.003751	0.008234	0	0.652255
Requested Disk	0.000527	0.001191	0	0.03766
Utilized Disk	0.000004	0.000023	0	0.000623
submission Rate	3.71	21.46	1	1956
task Length	21.55 (minutes)	69.79 (minutes)	1 (micro seconds)	24.67 (hours)

Table 3.4: Largest amount of each resource applied for de-normalization.

CPU	Memory (GB)	Disk (GB)
100% of a core of the largest machine CPU(3.2GHz)	4	1

ing utilized and requested amounts of resources that we conclude considering the plots in Figure 3.4. Table 3.3 also suggests that there is a high variability in submission rate and length of tasks. Here, the submission rate of tasks ranges from 1 to 1956 and the task length range is between 1 microseconds to more than 24 hours. High task submission frequency and short task lengths justify the inference of high variance of cloud workloads derived by Di et al. [44] (See Section 3.2).

As the reported resource request and utilization is normalized and the normalization is carried out in relation to the highest amount of the particular resource found on any of the machines [137]. In this context, to get a real sense of the data, we assume the largest amount of resources for each column as described in Table 3.4 and multiply each recorded data by the related amount (e.g for recorded memory utilization we have $Real_{util} = Recorded_{Util} * 4$). Then, the total resource utilization and requested amount are calculated for each cluster as discussed in the last section.

3.8 Characteristics of Task Clusters

X-means algorithm reports the existence of 18 clusters in the tasks. In this section, we go through the specifications of the task clusters in terms of the scheduling class, priority, and the average length of the tasks in each group (Table 3.5). The population comparison of the clusters is presented in Figure 3.5. To enable a better understanding of the characteristics of task clusters, Figure 3.6 summarizes Table 3.5 considering the similarities between task groups.

In Figure 3.6, task priority higher than 4 is considered "high". In addition, the average task length less than 1 and less than 5 hours are noted "short" and "medium" length respectively. The average task length higher than 5 hours is considered "long". Figure 3.6 shows that almost 78% of the tasks fall into the short length category. In addition, all long and medium length tasks have higher priorities and are less likely to be preempted. This logic is implemented in the Google cluster scheduler to avoid long tasks getting restarted

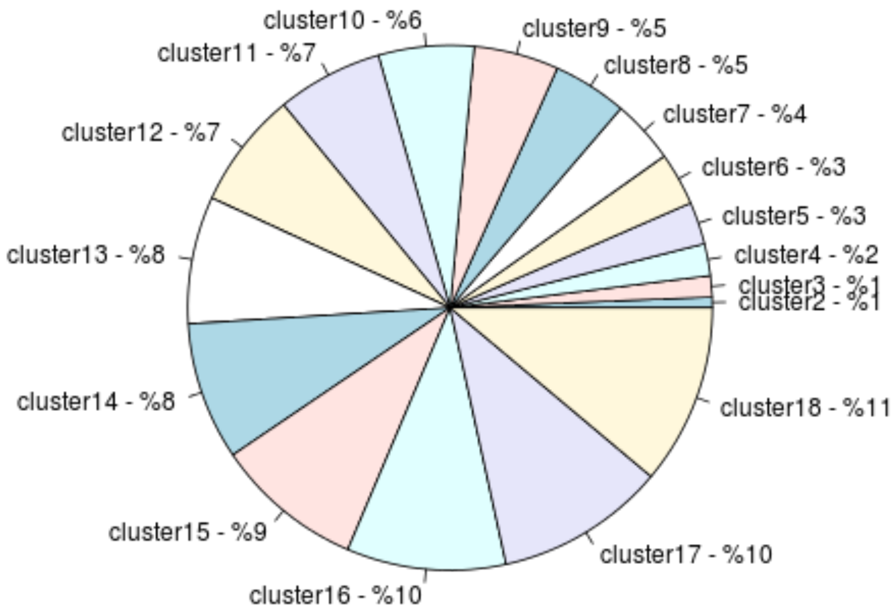


Figure 3.5: Population of tasks in each cluster. Clusters 15 to 18 are the most populated clusters. Since Cluster 1 population is less than 1%, it is not shown in the chart.

in the middle of execution, which leads to more resource wastage. Next, we describe the four meta-cluster task groups.

- **Short and high priority tasks (Cluster 2, 3, 13):** Tasks in clusters 2 and 3 are all from scheduling class 0. However, tasks in cluster 13 are from higher scheduling classes, which indicates that they are more latency sensitive than the tasks in clusters 2 and 3. Amongst these three clusters, cluster 13, with the average length of 38.66 minutes, has the longest average length.

- **Short and low priority tasks (Clusters 5, 6, 7, 10, 11, 12, 14, 15, 17, 18):** Comparing to others, this category includes the largest number of clusters. Cluster 7, with the average length of 56.82 minutes, has the longest tasks in this group. Considering the scheduling class, tasks in clusters 5, 6, 7, 10, 11, and 12 are all from scheduling class 0 while most of the tasks in clusters 14, 15, 17, and 18 are from scheduling class 1.

- **Medium and low priority tasks (Clusters 4, 8, 9, 16):** In terms of the average task length, Cluster 8, with 4.72 hours, has the longest length. Considering the scheduling class, tasks in Cluster 16 are more latency sensitive and probably belong

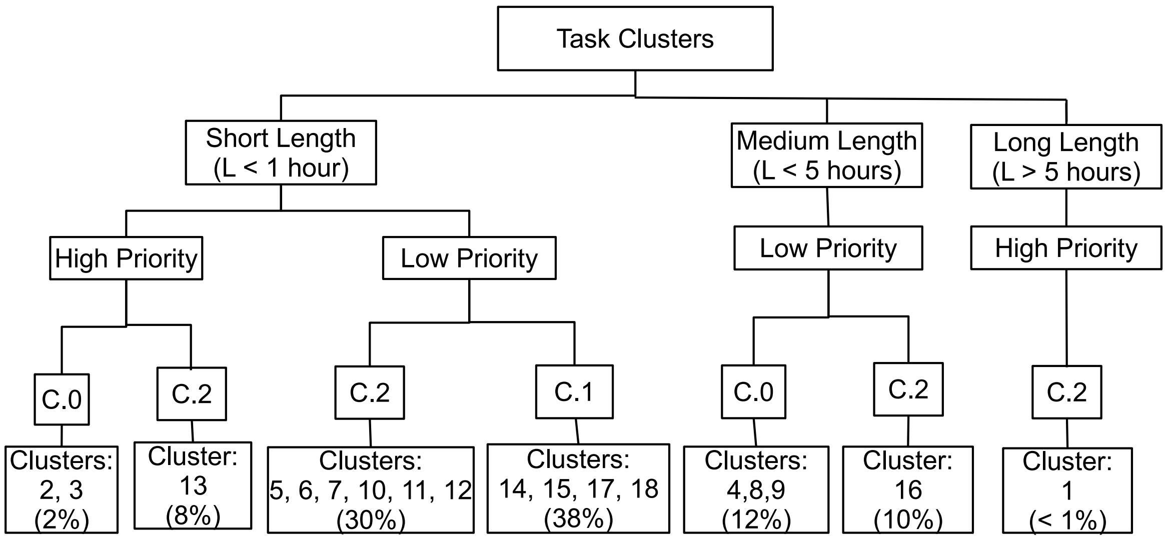


Figure 3.6: Clusters of tasks are categorized on three levels according to the average length, the priority, and the scheduling class (*C*) considering the statistics in Table 3.5.

to the production line while the tasks from the other three clusters are less latency sensitive.

- **Long and high priority tasks (Cluster 1):** Although Cluster 1 contains less than 1% of the tasks (Figure 3.5), this group has the highest priority tasks with the longest durations as shown in Table 3.5. Most of the tasks of the group have scheduling class 1, which shows they are less latency sensitive in comparison with tasks from higher scheduling classes.

Results of clustering allowed us to draw conclusions per cluster that help in the design of specific resource allocation policies for each cluster. For example, as depicted in Figure 3.6, tasks in Cluster 1 are the longest and have the highest priority. Therefore, one can conclude that the system assigns the higher priority to these long tasks so that if they failed, the system still has time to reschedule them. In contrast, as illustrated by Figure 3.6, the majority of tasks with short length have been given low priorities. This is because, in case of failure or resource contention, the system can delay their execution and still guarantee that they are executed in time.

In addition, as shown in Table 3.6, for task clusters with larger length, less usage variation is observed. For resource allocation policies, this makes the usage estimation of resources and predictions more accurate and more efficient, as less sampling data is

Table 3.5: Statistics of the clusters in terms of the scheduling class, priority and the average task length. The star sign (*) shows the dominant priority and scheduling class of the tasks in each group.

	Scheduling Class (%)			Priority (%)						Average Task Length
Cluster - 1	0	**1***	2	8	**9***	10	11			
	0.09	**99.76**	0.15	0.88	**99.10**	0.01	0.01			18.19(hrs)
Cluster - 2	**0***			**6***						
	100			**100**						6.38(mins)
Cluster - 3	**0***			**8***	9	10				
	100			**63.20**	36.76	0.04				5.7(mins)
Cluster - 4	**0***			**4***						
	100			**100**						1.04(hrs)
Cluster - 5	**0***			**4***						
	100			**100**						20.32(mins)
Cluster - 6	**0***			**4***						
	100			**100**						5.32(mins)
Cluster - 7	**0***			**0***	1	2				
	100			**97.02**	2.98	0.01				56.82(mins)
Cluster - 8	**0***	1		**0***	1	2	4	9		
	94.32	5.7		**83.44**	8.42	0.33	7.80	0.01		4.72(hrs)
Cluster - 9	**0***			0	**1***	2				
	100			36.55	**63.23**	0.23				1.47(hrs)
Cluster - 10	**0***			**0***						
	100			**100**						28.04(mins)
Cluster - 11	**0***			**1***	2					
	100			**99.2**	0.8					19.19(mins)
Cluster - 12	**0***			**0***						
	100			**100**						21.17(mins)
Cluster - 13	**2***	3		2	**4***	6	8	9	10	
	74.9	25.1		0.03	**30.97**	16.28	21.75	28.17	2.80	38.66(mins)
Cluster - 14	0	**1***	2	1	2	**4***	9			
	2.28	**97.62**	0.09	0.04	0.02	**99.49**	0.45			42.9(mins)
Cluster - 15	**1***			**4***	5	6				
	100			**97.67**	0.04	2.29				39.83(mins)
Cluster - 16	**2***	3		**0***	1	2	4	9		
	93.67	6.33		**67.61**	28.14	4.23	0.015	0.001		1.45(hrs)
Cluster - 17	**1***			**0***	1					
	100			**99.2**	0.8					27.59(mins)
Cluster - 18	**1***			1	**2***					
	100			2.7	**97.3**					20.49(mins)

Table 3.6: Virtual machine task capacity of each cluster for RRA, FqRA, AvgRA, MeRA, ThqRA, and URA resource allocation policies.

ClusterIndex	RRA	FqRA	AvgRA	MeRA	ThqRA	URA
Cluster - 1	15	29	34	32	38	136
Cluster - 2	20	33	39	40	43	125
Cluster - 3	20	16500	16500	16500	16500	500
Cluster - 4	41	33	43	38	56	38
Cluster - 5	28	32	48	46	56	139
Cluster - 6	8	51	88	77	117	250
Cluster - 7	80	46	64	53	62	143
Cluster - 8	73	45	48	47	48	150
Cluster - 9	41	95	104	104	107	154
Cluster - 10	6	7	9	7	8	250
Cluster - 11	12	91	92	99	105	188
Cluster - 12	11	28	53	46	71	1250
Cluster - 13	28	13	19	13	13	118
Cluster - 14	83	67	69	67	67	126
Cluster - 15	18	17	45	33	52	100
Cluster - 16	48	74	79	94	101	136
Cluster - 17	7	21	28	22	23	143
Cluster - 18	73	409	439	478	478	500

required while the prediction window can be widened. The opposite holds for clusters with smaller length: in these clusters, more variation is observed, and as a result prediction requires more frequent sampling and narrower time window.

3.9 Performance Evaluation

We discuss our experiment setup along with the results of experiments considering the aforementioned resource allocation algorithms.

3.9.1 Experiment Setup for Investigating Resource Allocation Policies

We discuss the setup of experiments that we conducted to evaluate our proposed approach in terms of its efficiency in task execution and power consumption. We studied the Google workload and grouped tasks according to their usage patterns utilizing clustering (Section 3.4). Then, the proposed system is simulated for each cluster and tasks are assigned to the corresponding virtual machine types during each processing window (one minute for the purposes of these experiments). The simulation runtime is set to 24 hours. Cluster resource usage and number of rejected tasks are reported for each cluster of tasks separately. Since virtual machine placement also affects simulation results, the same policy introduced in Section 3.9.1 is used in the Virtual Machine Provisioner component for all the proposed algorithms. In order to show the efficiency of our proposed architecture in terms of power consumption, the linear power consumption model is adopted for each of the running machines. The power consumption model is discussed in more details in the rest of this section.

Data Center Servers' Configuration

We define a data center with the three server configurations listed in Table 3.7. These types are inspired from Google data center and its host configurations during the studied trace period. Hosts in the Google cluster are heterogeneous in terms of the CPU, memory, and disk capacity. However, hosts with the same platform ID have the same architecture.

As mentioned in Section 3.2, there are three types of platforms in Google data center. In order to eliminate placement constraint for tasks, we have chosen the platform with the largest number of task submissions. The server architecture for our implementation is the same for all three types. As suggested by Garraghan et.al [63], servers in this platform are assumed to be 1022G-NTF (Supermicro Computer Inc.) inspired from the SPECpower_ssj2008 results [34].

Virtual Machine Placement Policy

The First Fit algorithm is applied as the placement policy for finding the first available machines for hosting newly instantiated VMs. The algorithm first searches through the running machines to find if there are enough resources available for the virtual machine. It reports the first running host that can provide the resources for the VM. If there is no running host found for placing the virtual machine, a new host is activated. The new host is selected from the available host list, which is obtained from the trace log and contains the hosts IDs along with their configurations. All the proposed algorithms have access to the same host list to make sure that the placement decision does not affect the simulation results.

Table 3.7: Available server configurations present in one of the platforms of the Google cluster [63].

Server Type	Number of Cores	Core Speed (GHz)	Memory (GB)	Disk (GB)	P_{idel} (W)	P_{max} (W)
Type1	32	1.6	8	1000	70.3	213
Type2	32	1.6	16	1000		
Type3	32	1.6	24	1000		

Server's Power Consumption Model

The power profile of the selected server[5] from SPECpower is used for determining the linear power model constants in Equation 3.5 [18]. The power consumption for processing tasks at time *t* is defined as the accumulative power consumed in all the active servers at that specific time. For each server, the power consumption at time *t* is calculated based on the CPU utilization and server's idle and maximum power consumption (Eq. 3.5). We focus on energy consumption of CPU because this is the component that presents the largest variance in energy consumption regarding its utilization rate [18].

$$P_n(t_i) = (P_{max} - P_{idle}) * n/100 + P_{idle} \tag{3.5}$$

3.9.2 Task Execution Efficiency of the Proposed Algorithms

After discussing the characteristics of extracted task groups, here we compare task execution efficiency of our proposed algorithms in terms of task rejection rate. Ideally, the percentage of tasks that need to be rescheduled should be as low as possible, since it results in delays in the completion of jobs. In addition to delays, the increase in task rejection rate increases resource wastage since computing resources (and energy) are spent on tasks that do not complete successfully and thus need to be later executed again. The rejection rate for each policy is presented in Figure 3.7.

Virtual machine capacity for each of the algorithms is shown in Table 3.6. In the URA policy, tasks are allocated based on the actual usage. Because of the gap between requested resources and the actual usage of tasks, in URA the VM task capacity is higher than in the other five algorithms. Therefore, in most of the clusters, RRA accommodates the least number of tasks in one virtual machine. Excluding RRA, FqRA has the smallest VM task capacity in comparison to the other four algorithms and excluding the URA

[5] 1022G-NTF (Supermicro Computer Inc.)

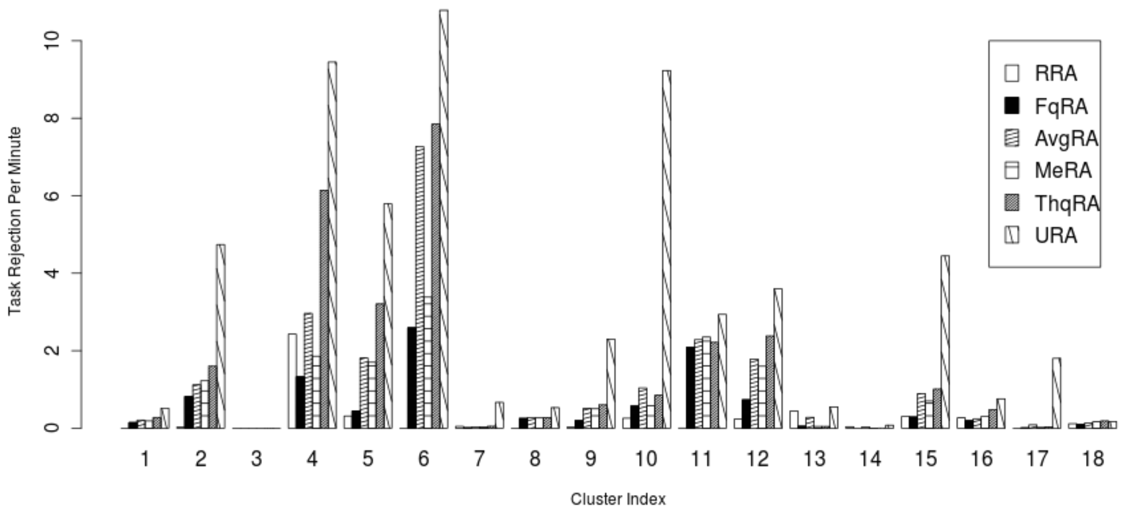


Figure 3.7: Task execution efficiency in the RRA, FqRA, AvgRA, MeRA, ThqRA, and URA policies. Efficiency is measured as the task rejection rate per minute.

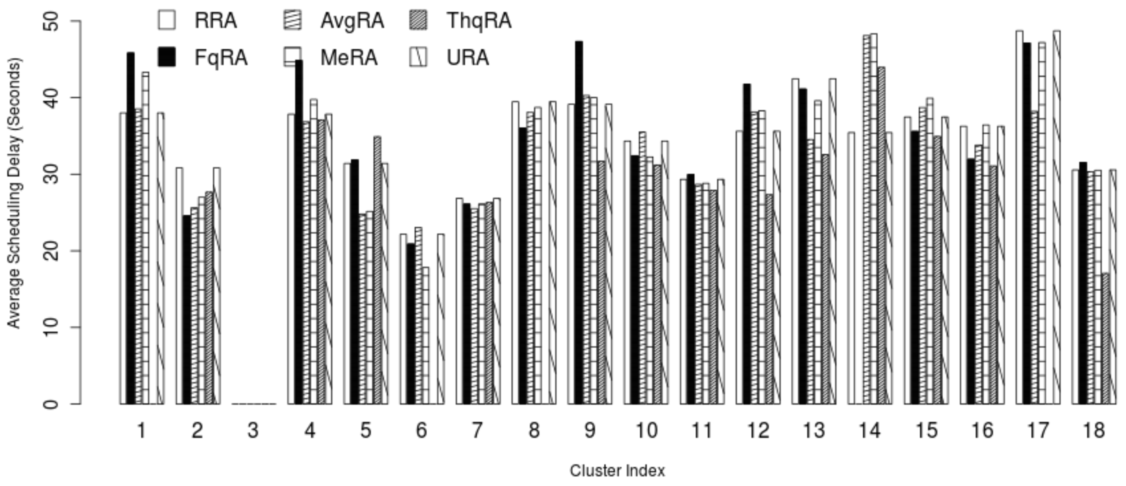


Figure 3.8: Average delay caused by applying the RRA, FqRA, AvgRA, MeRA, ThqRA, and URA policies. The delay is estimated by the time it takes for a specific task to be rescheduled on another virtual machine after being rejected.

policy, ThqRA has the largest amounts in terms of the task capacity.

Considering task rejection rate, the algorithms with larger amounts of VM task capacity have higher rejection rates. Therefore, in most clusters, URA has the highest rejection rate. However, the gap between rejection rates for FqRA, AvgRA, MeRA and ThqRA are almost negligible. As expected, RRA, with the lowest number of tasks in each virtual machine, incurs the least rejections during the simulation.

In addition to rejection rates, the delay caused in the execution of the tasks are re-

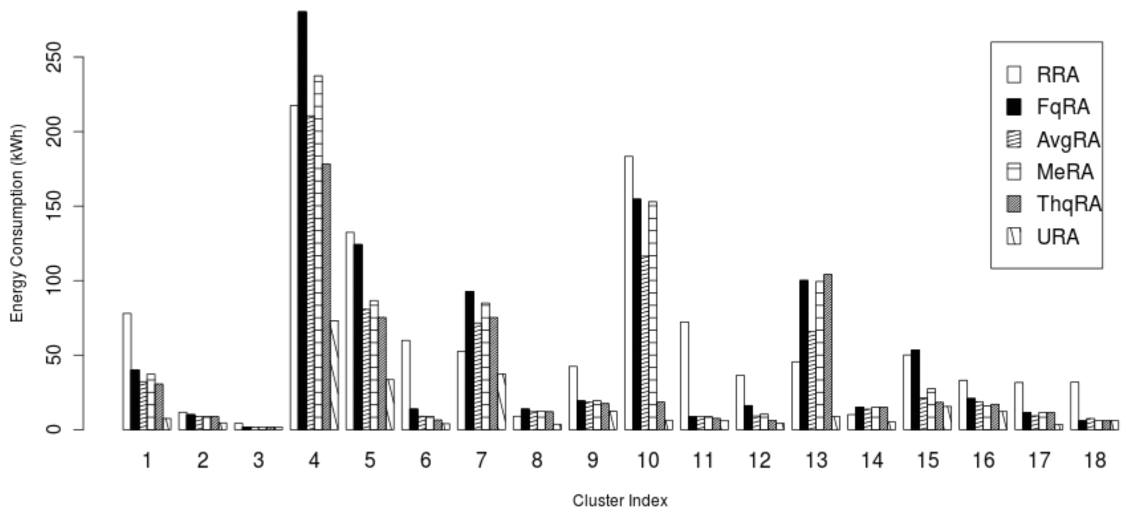


Figure 3.9: Energy consumption comparison of the *RRA*, *FqRA*, *AvgRA*, *MeRA*, *ThqRA* and *URA* policies. *URA* outperforms the other five algorithms in terms of the energy consumption and the average saving considering all the clusters.

ported for the proposed policies. This delay is extracted for rejected tasks that finish during the simulation time (24 hours). The delay t_d is equal to $t_f - t_g$ in which t_f is the time that the execution of the task is finished in our simulation and t_g is the desired finished time reported in the Google traces. In other words, t_d of a typical task is the time it takes for the task to start running after it is rejected. Figure 3.8 shows that the average delay for all the proposed algorithms is less than 50 seconds. This delay can be reduced via smaller processing window sizes. The processing window size in our case is assigned to one minute. Hence, tasks should wait in the killed task repository until the next processing window, so that they can get the chance to be rescheduled in another virtual machine.

3.9.3 Energy Efficiency of the Proposed Algorithms

The experiments presented in the previous section focused on the analysis of the performance of the assignment policies in terms of rejection rate and average delay. Since one of the goals of the proposed architecture is efficient resource allocation, which results in less energy consumption, in this section we analyze the policies in terms of their energy efficiency.

The power consumption incurred by servers are estimated using the power model presented in Equation 3.5. Figure 3.9 shows the amount of energy consumption (kWh) for the six applied resource allocation policies. In terms of energy consumption, URA on average outperforms RRA, FqRA, AvgRA, MeRA, and ThqRA by 73.02%, 59.24%, 51.56%, 53.22%, and 45.36% respectively, considering all the clusters. However, URA in most of the clusters increases the average task rejection rate and results in delays in task execution. Considering this, URA is the selected policy when tasks have low priorities and the delay in the execution is not a concern.

ThrdRA policy is the second most energy efficient algorithm, outperforming RRA, FqRA, AvgRA, and MeRA in average 34.41%, 25.11%, 7.42%, and 15.01% respectively. Apart from energy efficiency, this policy caused less task rejections in comparison with URA. Therefore, when task execution efficiency and energy are both important, this policy is the best choice. RRA in most clusters is the least energy efficient algorithm, although it caused less task rejections. Therefore, RRA can be applied for tasks with higher priorities.

AVgRA and MeRA have almost the same energy consumption for all the clusters. The task capacity of the VM in AvgRA and MeRA is based on the average and the median number of tasks that can run without causing any rejections. In most cases, the median and the average of our considered estimate (number of running tasks) are close to each other, therefore the difference in the energy consumption of AvgRA and MeRA is negligible.

3.9.4 Discussion

We investigated the problem of energy consumption resulted from inefficient resource allocation in cloud computing environments using Google cluster traces. We proposed an end-to-end architecture and presented a methodology to tailor virtual machine configuration to the workload. Tasks are clustered and mapped to virtual machines considering the actual resource usage of each cluster instead of the amount of resources requested by users.

Six policies were proposed for estimating task populations residing in each VM type. In the RRA policy, tasks are assigned to VMs based on their average requested resource. This policy is the baseline for our future comparison since it is solely based on the requested resources submitted to the data center. Resource allocation in the URA policy is based on the average resource utilization of task clusters obtained from historical data. In the other four policies, the assignment is based on the four estimates extracted from the virtual machines' usage logs from the URA policy. The extracted estimates are average, median, first and third quantile of the number of tasks that can be accommodated in a virtual machine without causing any rejections. Compared to RRA, the other five policies show up to 73.01% improvements in the data center total energy consumption.

Comparing the results from the six studied policies the following conclusions can be derived:

- By analyzing cloud workload, cloud providers and users can tailor their virtual machine configurations according to their workload usage patterns. This would result in less resource wastage and be cost effective on both consumers and cloud providers side. Customized VM types are now offered by Google[6] cloud.

- Results demonstrate that utilization of historical data regarding actual resource utilization can help in decreasing the amount of hardware consumption and consequently reduce the energy usage of cloud data centers. However, it is advisable to analyze VM's resource consumption instead of task clusters when it comes to the amount of rejected tasks and violations.

- Customized VM sizes are beneficial for workloads that are not a good fit for available VM sizes or for workloads that require more computing resources [6].

Based on the above insights, next we evaluate the utilization of customized VM sizes compared to fixed VM configurations. In addition, we explore the Container as a Service (CaaS) cloud model, in which users request the execution of containers, which are deployed inside virtual machines. We explore this model because it represents more closely the architecture used by Google when generating the Google traces utilized in these experiments. A key difference between Google architecture concerns the envisioned deployment model: Google traces related to a private cloud accessible only by Google employees. Our approach targets any deployment model, and therefore an extra layer of

[6]`https://cloud.google.com/compute/docs/instances/creating-instance-with-custom-machine-type`

performance isolation between tenants, enabled by virtualization, is also included in the architecture. *Notice that this is exactly the method that public cloud providers utilize to offer CaaS: containers on top of VMs rather than on bare hardware (Amazon ECS*[7] *and Container Engine*[8]*).*

3.10 Efficient VM Sizing for CaaS

As we mentioned a new type of service—called Containers as a Service (CaaS)—has been introduced by Google[8] and AWS[7]. To reduce the energy consumption of CaaS, one may choose virtual machine consolidation, Dynamic Voltage and Frequency Scaling (DVFS), or both of them combined. However, these efforts would be in vain if VM sizes are not customized to better support deployed containers. For example, as shown in Figure 3.1, the size of *VM B* in comparison to *VM A* is not container-optimized. As a result, there is resource wastage that results in inefficiency in terms of energy consumption regardless of how effective and energy efficient is the VM consolidation technique in place. In other words, as we discussed in Chapter 2, VM consolidation is limited by other resources such as memory (Figure 2.5), hence for improving the resources utilization of the servers, resources of virtual machines should also be allocated and utilized efficiently.

3.10.1 Extended System Model

In order to incorporate the CaaS cloud model, we extended our studied system model in Section 3.3 and added one more layer to it.

Cloud Model: The cloud model explored in this chapter contains three layers:

- **Infrastructure layer:** This layer contains physical servers that are partitioned to virtual machine through the next layer;
- **Virtualization layer:** The virtualization technology is taken into consideration as it improves the utilization of resources by sharing them between virtual machines;

[7] https://aws.amazon.com/ecs/
[8] https://cloud.google.com/container-engine/

- **Container layer:** Containers run on virtual machines and all share the same Linux kernel as the OS. The combination of these technologies (i.e., container and virtualization) has been recently introduced in Google Container Engine[9], which uses the Open Source technology Kubernetes[10].

3.10.2 Extended Architecture

We utilize the architecture proposed in Section 3 (Figure 3.2) for selecting the right size of virtual machines that hosts containers running users' tasks. However, the task mapper component in the execution phase is substituted by another component named "Container Mapper". Similar to *Task Mapper*, the Clustering results from the Task Classifier are sent to the *Container Mapper*. Firstly, this component maps each task to a suitable container. Then, as it is depicted in Algorithm 1, based on the available resources in the running virtual machines and the available VM types in the *VM Types Repository*, this component estimates the number and type of new virtual machines to be instantiated to support the newly arrived tasks. Apart from new VM instantiation when available VMs cannot support the arriving load, this component also reschedules rejected tasks that are stored in the rejected task repository to the available virtual machines of the type required by the VM (if any). This component prioritizes the assignment of newly arrived tasks to available resources before instantiating a new virtual machine.

Modified VM Sizing Strategy:

We modified the sizing strategy for VMs in Section 3.5 to take into account the infrastructure's owner limits in terms of the minimum number of VMs that can be accommodated on each server. For each group (cluster) of tasks, in order to determine the virtual machine's parameters, the average amount of resources required per hour for serving the user requests during the 24 hours observation period is estimated. Thus, the amount of CPU required per hour (CPU_h) is defined as follows:

$$CPU_h = \overline{nT} * \overline{CPU}_u \tag{3.6}$$

[9]Google Container Engine: https://cloud.google.com/container-engine/docs/
[10]Kubernetes: http://kubernetes.io/

where $\overline{CPU_u}$ is the average amount of CPU usage on an hourly basis and $\overline{nT_h}$ is the average number of tasks per hour that are present and running in the system.

When the average amount of required CPU per hour is estimated, a limit should be set for the maximum amount of the CPU that a virtual machine can obtain. This amount should be set according to the underlining infrastructure. For example, if the servers have 32 cores of CPU and the provider wants to host at least two virtual machines per server, then each VM can obtain at most 16 cores. If we assume that the largest virtual machine in the system would have less than 16 cores, then if the CPU_h is above 16, the load should be served by more than one virtual machine. It has to be divided by the first integer n found between 2 to 9 in a way that the residual of CPU_h/n is zero (For example if the CPU_h is equal to 21, n would be 3, which results in 3 VMs with 7 cores each). In other words, n is the number of virtual machines with $VM_{CPU} = CPU_h/n$ which is enough for serving the requests for every hour. Moreover, if the CPU_h is bellow the virtual machine's maximum CPU limit, then one virtual machine would be enough to serve the requests and n is equal to 1.

Then, for defining the VM memory size, Equation 3.7 is used.

$$VM_{memory} = \frac{\overline{nT} * \overline{memory_u}}{n} \tag{3.7}$$

The VM's task capacity VM_{tc} is calculated based on Equation 3.8 where $\overline{t_{U_i}}$ is the average resource usage during the observed period.

$$VM_{tc} = min(VM_1/\overline{t_{U_1}}, ..., VM_i/\overline{t_{U_i}}), i = \{CPU, memory\} \tag{3.8}$$

Thus, for each cluster, the VM type is defined in terms of CPU_h and VM_{memory}. Once VM Types are determined, they are stored in the VM types repository, so future selection of better matches for required containers hosting the tasks are chosen based on such defined types. The effectiveness of this strategy for selection of VM Types based on clustering of tasks is evaluated in the next sections.

3.10.3 Experiment Setup for Investigating VM Sizing Efficiency

In this section, we discuss the experiments that we conducted to evaluate the efficiency of our proposed VM sizing approach in terms of its efficiency in task execution and power consumption. The data set used for these experiments is also derived from the second day of the Google cluster traces [137]. In order to get a real sense of the data, we assume the same amounts mentioned in Section 3.9.1 as the largest amount of resources including CPU, memory, and disk.

As we discussed, the Google cluster hosts are heterogeneous in terms of the CPU, memory, and disk capacity. However, hosts with the same platform ID have the same architecture. Like our previous experiment setup in Section 3.9.1, in order to eliminate the placement constraints and decrease placement complexity, only the tasks scheduled on one of the three available platforms are considered. The configurations of the simulated data center servers are also inspired by Google data center during the studied trace period. However, in order to check the applicability of our proposed approach, we utilized the tasks submitted to the platform with the second highest submission rate in the second day of the traces. As suggested by Garraghan et.al [63], the servers in this platform are PRIMERGY RX200 S7. For the PRIMERGY platform, the tasks scheduled during the second day of the traces are all scheduled on one server type with 32 cores of CPU (3.2 GHz), 32 GB of memory and 1 TB of disk. The power profile of this server type is extracted from the SPECpower_ssj2008 results [34] reported for PRIMERGY. This power profile is then used for determining the constants of the applied power linear model presented in Equation 3.5.

In our experiments, the proposed system is simulated and the tasks are assigned to containers. Then, these containers are hosted in the corresponding virtual machine types during each processing window (one minute for the purposes of these experiments). The simulation runtime is set to 24 hours. The number of rejected tasks is reported for each experiment separately. Since the virtual machines placement also affects the simulation result, First-Fit placement policy is used for all of the experiments. This placement algorithm selects the first running host that can provide resources for the VM and if no running host is found for placing the VM, then a new host will be turned on.

Output metrics of interest are number of instantiated VMs, task rejection rate, and

Table 3.8: Virtual machine configurations for 18 clusters.

VM Type	Number of Tasks	vCPU	Memory (GB)	VM Type	Number of Tasks	vCPU	Memory (GB)
TYPE 1	142	14	17.93	TYPE 10	418	8	14.8
TYPE 2	79	1	2.82	TYPE 11	471	9	28.98
TYPE 3	261	3	11.70	TYPE 12	87	2	4.11
TYPE 4	292	2	2.39	TYPE 13	439	1	3.68
TYPE 5	1836	1	5.35	TYPE 14	185	3	22.48
TYPE 6	107	3	8.1	TYPE 15	78	1	2.72
TYPE 7	70	2	3.21	TYPE 16	220	3	10.12
TYPE 8	585	1	0.93	TYPE 17	254	1	14.14
TYPE 9	336	1	2.06	TYPE 18	1200	1	1.52

Table 3.9: Virtual machine specifications of RFS and the selected Amazon EC2 instances.

RFS				Amazon EC2			
VM Type	Number of Tasks	vCPU	Memory (GB)	VM Type	C_{max}	vCPU	Memory (GB)
TYPE 1	452	1	5.42	m3.large	150	1	7.5
TYPE 2	696	9	16.1				
TYPE 3	273	3	5.32	m3.medium	150	1	3.75
TYPE 4	1639	11	24.29	t2.medium	266	2	4
TYPE 5	357	1	3.29	t2.small	133	1	2

the energy consumption (Subsection 3.10.6). In order to ensure the quality of service, task rejection rate is defined as the number of rejected tasks in each 1-minute processing window.

3.10.4 Feature set selection

we explore the factors that affect the resulting virtual machine sizes (types). The number of VM types is directly affected by the number of task clusters (k), which is estimated via applying X-means on the selected feature set. The following approaches are considered in the investigation of the effect of feature selection on virtual machine sizes:

- **Whole Feature Set (WFS) approach:** In this approach, the whole feature set listed in Section 3.4.1 is considered as the input of the X-means algorithm, which resulted in 18 clusters of tasks. The virtual machine sizes are defined following the procedures in Section 3.10. The obtained VM sizes are listed in Table 3.8. As we mentioned in Section 3.10, 10 GB of storage is assigned to all VM types in order to have enough space for the OS installed on each VM.

- **Reduced Feature Set (RFS) approach:** By assigning the same amount of storage, WFS approach leads to the observation that the most effective parameters in VM size selection for this specific workload are the average CPU and memory utilization. Therefore, for the RFS approach, the clustering feature set is reduced to these two main features. The application of the new feature set as the input of X-means resulted in 5 clusters, which consequently results in 5 different VM sizes (shown in Table 3.9). Between these two approaches, the one that could save more energy

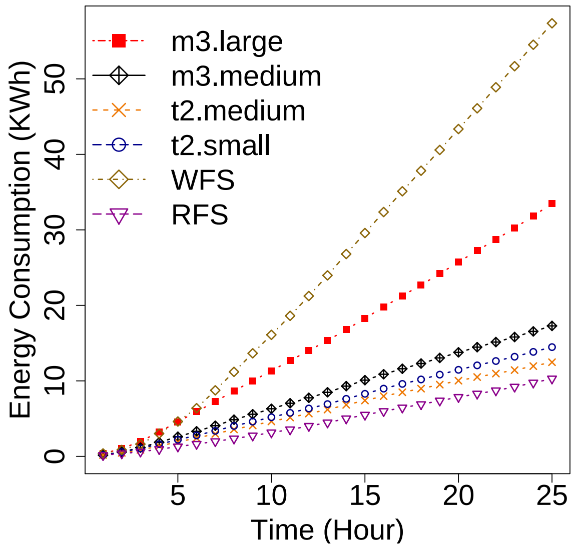


Figure 3.10: Energy consumption of the data center for the usage-based fix VM size approach versus RFS and WFS

via reducing the number of servers has the more efficient virtual machine sizes. As Figure 3.10 shows, RFS results in 81% less energy consumption in comparison to WFS. This is because RFS causes less resource fragmentation and that leads to less number of servers. Therefore, it can be concluded that tailoring the size of virtual machines to container requirement is crucial, however it has to be done in such a way that resource fragmentation is avoided. Apart from the energy perspective, comparing to WFS approach, RFS results in 86% and 77% less instantiated VMs and rejection rate respectively.

3.10.5 Baseline scenarios

In order to investigate the effect of defining the virtual machine sizes according to the workload (container optimized), in these experiments we consider scenarios where VM sizes are fixed. In this respect, we consider the existing Amazon EC2 instances listed in

Table 3.9. In these scenarios, the scheduler assigns containers to only one specific VM size (e.g t2.small instance) considering the following approaches:

- **Request-based resource allocation approach:** In the request-based resource allocation approach, tasks are given the exact amount of resources requested and specified by the user while submitted. Therefore, in this approach containers are packed in the virtual machines according to their containing task specifications, as submitted by users.

- **Usage-based resource allocation approach:** In this approach, the maximum number of containers (c_{max}) that can be hosted in one VM instance is specified in the pre-execution phase leveraging the average resource utilization of the tasks. c_{max} is reported for each of the Amazon instances in Table 3.9. This approach is included in the baseline scenarios, since the concept of utilizing the usage of the tasks is also taken into account in our VM size selection technique.

$$c_{\mathbf{max}} = min(\mathrm{VM}_r/\overline{\mathrm{Usage}_r}), r = \mathrm{CPU, Memory, Disk} \tag{3.9}$$

3.10.6 Experiment Results

Results obtained with the utilization of aforementioned approaches are compared considering three different matrices. The first matrix, which shows the main effect of our proposed VM size selection technique, is the energy consumption matrix obtained through Equation 3.5. Figure 3.10 shows the energy consumption for the baseline scenarios considering the usage-based approach and the proposed VM size selection technique. The data center energy consumption for the baseline scenarios considering the requested-based approach is also plotted in Figure 3.11. Because users overestimate resource requirements, the energy consumption is improved in the usage-based approach by almost 50% comparing to the requested-based approach in the baseline scenarios, for all of the fixed VM sizes. However, RFS still outperforms all of other approaches in terms of the data center energy consumption. For the usage-based baseline scenarios, RFS shows 33%, 44%, 24%, and 70% less energy consumption on average compared to t2.small, t2.medium, m3.medium, and m3.large VM sizes respectively.

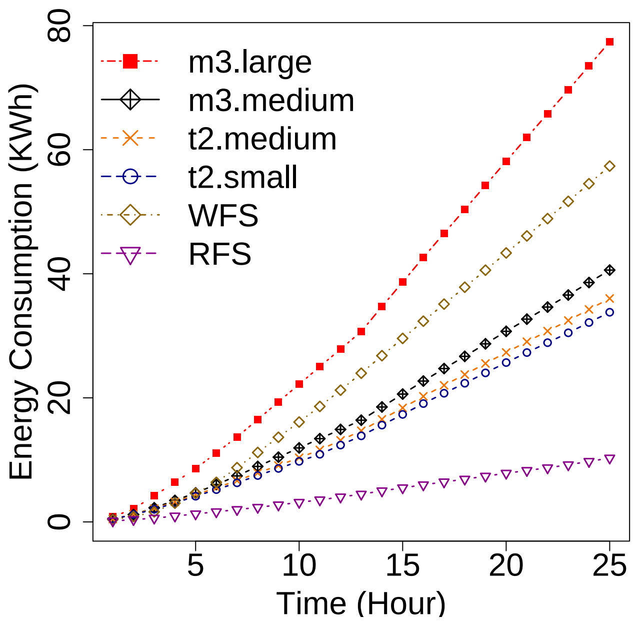


Figure 3.11: Energy consumption of the data center for the request-based fix VM size approach versus RFS and WFS.

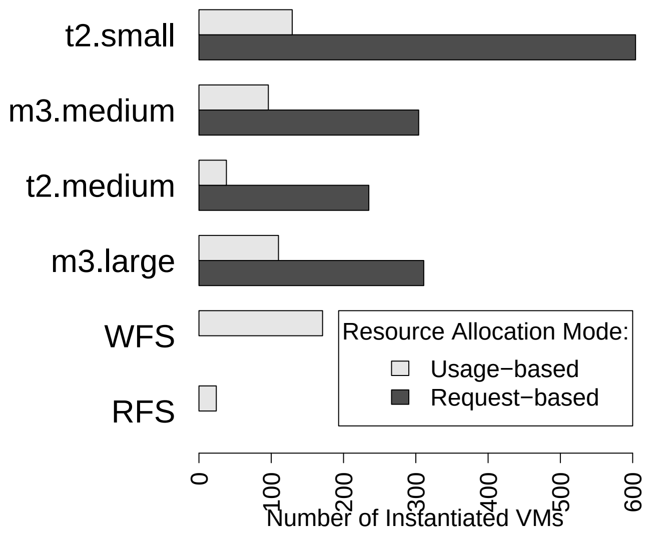


Figure 3.12: Number of instantiated virtual machines for the applied approaches.

For the second comparison matrix, we consider the number of instantiated virtual machines (n_{VM}). The increase in n_{VM}, results in more virtualization overhead and also longer delays in scheduling because of the VM instantiation delay. Results show that RFS can execute the workload with the smallest n_{VM}. It outperforms usage-based baseline scenarios by 82%, 75%, 37%, and 86% less instantiated VMs for the t2.small, t2.medium, m3.medium, and m3.large VM sizes respectively (Figure 3.12).

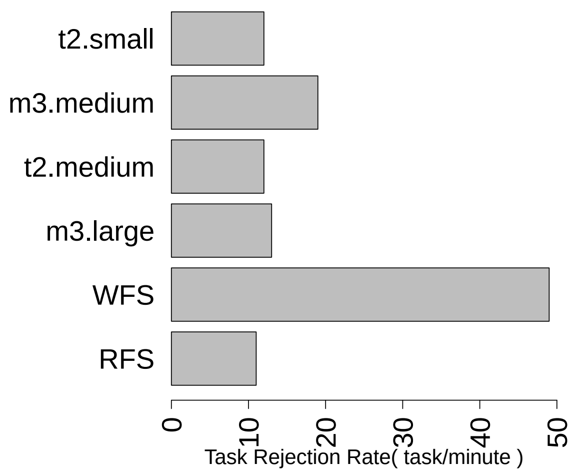


Figure 3.13: Task rejection rate for WFS, RFS and the fixed VM sizes considering the usage-based approach

In order to ensure the quality of service, the task rejection rate is considered as the third comparison matrix. Task rejection rates for the usage-based baseline scenarios, WFS, and RFS approaches are shown in Figure 3.13. As in previous cases, RFS VM size selection approach outperforms the usage-based baseline scenarios by 8%, 42%, 8%, and 15% less task rejection rate for the t2.small, t2.medium, m3.medium, and m3.large VM sizes respectively. It is worth mentioning that, because of the over-allocation of resources for the requested-based baseline scenarios, the task rejection rate is equal to zero for all of the VM types.

In summary, the experiments show that our VM sizing technique combined with workload information saves a considerable amount of energy with minimum task rejections. This implies that studying the usage patterns of applications would improve the utilization of resources and consequently would reduce the energy consumption of data centers. However, the study of the workload should be accurate in a way that it would not result in more resource wastage as it is shown in RFS approach.

3.11 Conclusions

In this chapter, we investigated the problem of inefficient utilization of data center infrastructure resulted from users' overestimation of required resources on the virtual machine level. To address the issue, we presented a technique for finding efficient virtual machine sizes for hosting tasks considering their actual resource usage instead of users estimated amounts.

We clustered the tasks and mapped them to the virtual machines according to their actual resource usage. Then, we proposed six policies for estimating task populations residing in each VM type. In the baseline scenario (RRA policy), we assigned the tasks to the VMs considering their average requested resource. While in the second resource allocation policy (URA), the tasks are assigned based on their average resource utilization of task clusters obtained from historical data. The assignment of tasks in the other four policies is based on the virtual machines' usage logs from the URA policy. These estimates include average, median, first and third quantile of the number of tasks that can be accommodated in a virtual machine without causing any rejections. The experiment results demonstrate that considering the resource usage patterns of the tasks improves the energy consumption of the data center.

Further, we extended our proposed technique to incorporate the CaaS cloud service model and investigate the efficiency of our VM sizing technique considering the clustering feature set. We considered baseline scenarios in which virtual machine sizes are fixed. Again, due to user overestimation of resources, the usage-based approach (where VM sizes are chosen based on actual requirements of applications rather than the amount requested by users) outperforms the requested-based approach by almost 50% in terms of the data center average energy consumption. In addition, the results demonstrate that over analyzing the workload (18 clusters resulted from RFS policy) would result in resource wastage while right workload analysis (8 clusters resulted from WFS policy) improves the resource utilization and consequently decrease the data center energy consumption.

In addition to determining efficient VM sizes, dynamic container consolidation also affects the efficiency of VM's resource utilization. In order to evaluate and compare the performance of the resource management algorithms for containerized clouds, we require an environment that supports scalable and repeatable experiments. Therefore, in the next chapter, we present our developed simulator, which provides support for modeling and simulation of containerized cloud environments and for exploring efficiency of container consolidation algorithms.

Chapter 4

Modeling and Simulation of Containers in Cloud Data Centers

Containers are increasingly gaining popularity and becoming one of the major deployment models in cloud environments. To evaluate the performance of scheduling and allocation policies in containerized cloud data centers, there is a need for evaluation environments that support scalable and repeatable experiments. Simulation techniques provide repeatable and controllable environments and hence they serve as a powerful tool for such purpose. This chapter introduces ContainerCloudSim, which provides support for modeling and simulation of containerized cloud computing environments. We developed a simulation architecture for containerized clouds and implemented it as an extension of CloudSim. We have described a number of use cases to demonstrate how one can plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance. Our system is highly scalable as it supports simulation of large number of containers, given that there are more containers than virtual machines in a data center.

4.1 Introduction

DUE to the elasticity, availability, and scalability of its on-demand resources, cloud computing is being increasingly adopted by businesses, industries, and governments for hosting applications. As discussed in Chapter 2, in addition to traditional cloud services, namely Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), recently a new type of service—Containers as a Service (CaaS)—has been introduced . Containers share the same kernel with the host, hence they are defined as lightweight virtual environments compared to VMs that provide a

This chapter is derived from: **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros , and Rajkumar Buyya, "An Environment for Modeling and Simulation of Containers in Cloud Data Centers", *Software: Practice and Experience (SPE)*, John Wiley & Sons, Ltd, USA, 2016 . [Online]. Available: http://dx.doi.org/10.1002/spe.2422.

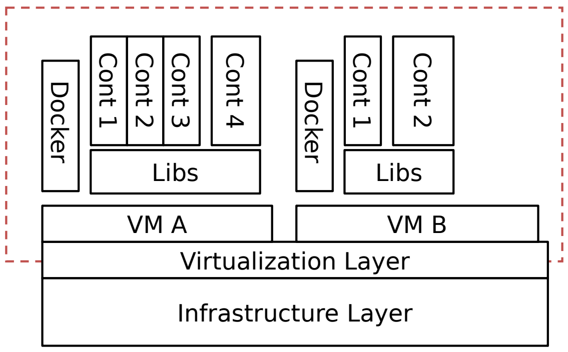


Figure 4.1: The virtual environment modeled in ContainerCloudSim.

layer of isolation between workloads without the overhead of hypervisor-based virtualization. CaaS providers, such as Google and AWS, argue that containers offer appropriate environment for semi-trusted workloads, while virtual machines provide another layer of security for untrusted workloads.

Resource management policies to ensure Quality of Service (QoS), avoid energy wastage and resource fragmentation are an integral part of cloud systems. Innovating and comparing resource management strategies require environments that facilitate the design of experiments while making them repeatable and accurate. Simulators are useful tools to build such evaluation environments in the cloud context [173]. They are particularly helpful at early stages of research to identify and eliminate ineffective polices or when accessing large scale distributed infrastructure is costly and not possible. Testing and evaluating resource management policies in the first verification stage in a production environment is both risky and costly. In this respect, a number of simulation tools are developed for evaluation of algorithms that are specifically designed for cloud computing environments. Although containers are going to be one of the dominant application deployment models in the cloud, most of the simulators consider VMs as the building blocks of the virtualized cloud data centers. To the best of our knowledge, no simulators introduced modeling for containerized cloud environments. Therefore, this chapter presents a simulation environment, named "*ContainerCloudSim*", for studying resource management techniques in CaaS environments.

ContainerCloudSim is developed as an extension of the ClouSim simulation toolkit [26]. It provides an environment for test and evaluation of resource management techniques such as container scheduling, placement, and consolidation of containers. As depicted in

Figure 4.1, *ContainerCloudSim* uniquely enables researchers to consider resource management techniques for both virtualization types including the system level virtualization/ containers (Figure 2.2) and Operating System level virtualization/VMs (Figure 2.6) side by side. For the *Virtual Machine* type, applications execute inside virtual machines and for the CaaS model, applications execute inside containers while the containers are placed in virtual machines. The proposed simulator models a container migration by stopping the container on the source host and starting it with a realistic delay on the destination host, which closely resembles current containerized environments. Moreover, *ContainerCloudSim* offers an environment to evaluate various (power-aware) resource management algorithms by providing diverse power models in a data center.

4.2 Related Work

As discussed earlier, simulation environments can speed up the development process of theoretical research by allowing repeatable experiments in a controllable environment [149]. Simulators enable the study of the effect of one parameter on the objective of research while keeping the other parameters controlled, which might be difficult or sometimes impossible to achieve in a real world scenario. Considering the significant benefits of simulation for cloud computing environments, a number of simulators with various objectives were developed [49]. The simulators differ in considered performance metrics, supported applications, and whether they consider power consumption or not.

MDCSim [108] is a commercial comprehensive and scalable simulation toolbox that is used for in-depth analysis of multi-tier data centers. It models the underlying hardware characteristics of data center components and estimates the power consumption of data centers. Throughput and response time are considered as performance metrics and the topology of the data center is supplied as a directed graph by the MDCSim network package. MDCSim helps cloud users to examine different resource configurations to improve the performance of web applications while keeping the power consumption low. Likewise, **GDCSim** [74], as a simulation tool, is especially developed to help service providers to test the impact of different data center physical designs and resource management algorithms on power consumption before deployment. GDCSim is extensible so that the user can add new models of power consumption, resource management, and cooling. Similar to GDSim[74] and MDCSim [108], *ContainerCloudSim* also enables users to have an estimate of the data center power consumption by using the available built-in or user-defined power models.

CloudSim is developed as an extensible cloud simulation toolkit that enables modeling and simulation of cloud systems and application provisioning environments [26]. This toolkit provides both system and behavior modeling of cloud computing components such as virtual machines (VMs), data centers, and users. It also enables the evaluation of resource provisioning policies in a cloud computing environment. The generic application provisioning techniques implemented in CloudSim can be extended easily with limited effort. It also supports modeling and simulation of both single and inter-networked clouds (federation of clouds) and exposes custom interfaces for implementing policies and VM provisioning techniques. In *ContainerCloudSim*, CloudSim is extended to enable modeling a containerized cloud environment that is not currently supported by CloudSim or any of its extensions.

Simulation Program for Elastic Cloud Infrastructures (**SPECI**) [148] is a simulation toolkit that focuses on scalable design of cloud data centers. In addition, it is capable of testing failure and recovery mechanisms. This enables exploring aspects of scalability along with performance properties of future data centers. The objective of SPECI is simulating the performance and behavior of data centers having the size and middleware design policy as input.

GroudSim [125] is a Java-based simulation toolkit especially designed for simulating scientific applications execution both on Grid and cloud infrastructures. GroudSim provides users with basic statistics and analysis after the simulation. It also supports modeling of computational and network components, job submissions, and file transfers. Similar to SPECI [148], failures in GroudSim can be modeled and integrated with background load, and cost models. *ContainerCloudSim* can also be extended to incorporate the modeled application and machine failures.

Data center Simulator (**DCSim**) [152] is an extensible simulation framework developed aiming at investigating dynamic resource management techniques in Infrastructure as a Service cloud deployment model. The key features introduced in DCSim include a multi-tier application model and the modeling of interactions and dependencies between virtual machines (VMs). VM replication is another feature available in DCSim and is utilized for handling increases in the workload.

GloudSim [42] is developed as a distributed cloud simulator based on the second version of the Google traces considering virtualization technology (VMs). GloudSim introduced three main features. The first feature is the ability to emulate resource utilization of reproduced jobs as closely as possible to the real values in the trace. The second feature is the simulator's ability to precisely emulate the different event types such as kill/evict based on the trace. Finally, the simulator can emulate more complex cases beyond the original trace investigating the challenges in resource management in cloud computing environment. GloudSim can also reproduce check-pointing/restart events considering the Google trace leveraging Berkeley Lab Checkpoint/Restart (BLCR) tool. Like GloudSim [42], *ContainerCloudSim* also provides the support for incorporating the cloud data centers' resource utilization traces. Currently, the workload models of PlanetLab [127] is utilized as the container's usage data.

iCanCloud [124] is a simulation tool that aims to predict the trade-offs between cost and performance of a set of applications executed in a cloud data center. iCanCloud can be used by different users including basic cloud users and distributed application developers. The simulation platform of iCanCloud provides a scalable, fast, and easy-to-use tool helping users to obtain results quickly considering their budget limits. iCanCloud is based on SIMCAN[1] and provides a graphical user interface that enables users to execute the experiments. In addition, it models network communication between machines and supports parallel simulations, hence an experiment can be executed across multiple machines.

In **CloudAnalyst**, Wickremasinghe et al. [164] extended CloudSim to enable applications workload description including the number of users, data centers, and cloud resources along with the location of both users and data centers. CloudAnalyst can be used by application developers or testers to determine the best strategic allocation of resources among the available cloud data centers. Data centers can be selected strategically considering the application workload and the available budget. Like iCanCloud [124],

[1] http://www.arcos.inf.uc3m.es/~simcan/

CloudAnalyst also provides a graphical interface which simplifies the process of building a number of simulation scenarios.

In **TeachCloud** [85], CloudSim is extended with a model for MapReduce application and an integrated comprehensive workload generator called *Rain*. It enables experiments with various cloud components including processing elements, storage, networking, and data centers. Like the two aforementioned simulators [124,164], it also supports a graphical interface that enables building and implementing customized network topologies. It also provides a VL2 network topology model. TeachCloud, as a comprehensive and easy-to-use tool, can be utilized as a educational tool that allows students to conduct experiments in a cloud system.

CDOSim [56] is developed extending CloudSim to model response times, SLA violations, and costs of a cloud deployment option (CDO). CDOSim is oriented towards the cloud user's side instead of investigating the issues on the cloud provider side. The user behavior can be supplied through workload profiles extracted from production monitoring data. CDOSim can simulate any application as long as it can be modeled following the Knowledge Discovery Meta-Model (KDM) [134]. The notion of million instructions per second (MIPS) unit in CloudSim is refined to the mega integer plus instructions per second (MIPIPS) unit. CDOSim is integrated in the cloud migration framework CloudMIG [60] and is available as a plug-in for the CloudMIG Xpress tool. Contrary to CDOSim [56], *ContainerCloudSim* is mainly focused on the provider's side.

In **NetworkCloudSim**, Garg et al. [62] extended CloudSim to provide a scalable network and generalized application model. It supports applications with communicating elements including Message Passing Interface (MPI) and workflows. A network flow model is designed for cloud data centers and bandwidth sharing is made possible. The extension is developed in such a way that users can modify the network topology simply by modifying a configuration file.

Contrary to NetworkCloudSim [62], **GreenCloud** [102] is developed as a packet level simulator on top of the NS2 simulator [1].

GreenCloud is specifically designed to investigate power management schemes to achieve an energy efficient data center. These schemes include both voltage and frequency scaling, and dynamic shutting down of network and compute components. It

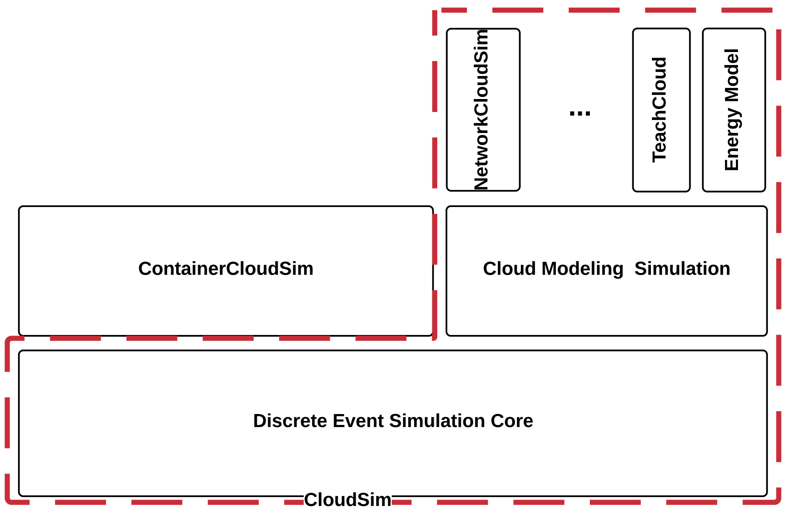


Figure 4.2: *ContainerCloudSim* relations to the CloudSim ecosystem.

enables the capture of details of the energy consumption of data center's computing and network components. It also considers the packet-level communication patterns between network components. Like CloudSim, ContainerCloudSim can also be extended to incorporate network components and the communications between containers.

In summary, in comparison to our proposed ***ContainerCloudSim***, existing simulators do not support modeling and simulation of containers in a cloud environment. They primarily focus on system level virtualisation with virtual machine as the fundamental component [25,42,56,62,74,85,102,124,125,152,164]. In comparison to other CloudSim extensions [25,62,85,164], *ContainerCloudSim* has been developed on top of the CloudSim core as depicted in Figure 4.2.

4.3 CaaS modeling requirements

Since Container as a Service (CaaS) is a newly introduced service in public cloud computing environments, there is still a lack of defined methods and standards that can efficiently tackle both application-level and infrastructure complexities. However, as container technologies mature and are broadly adopted, research in this topic will also emerge, proposing new algorithms and policies for containerized cloud environments. To reduce the development time of these new approaches, there is need for an environment that provides functionalities to enable robust experiments with various setups

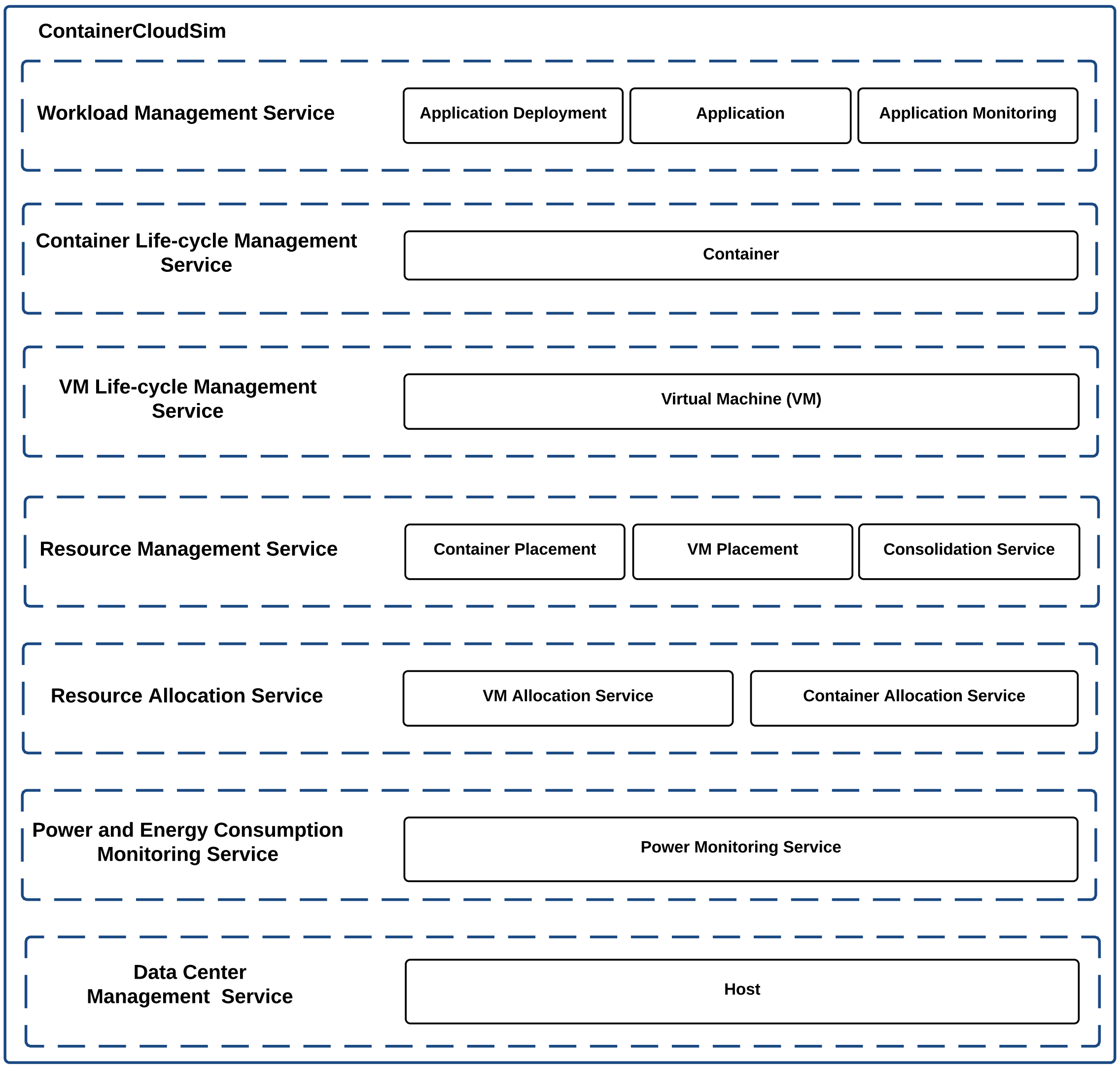

ContainerCloudSim
Workload Management Service
Application Deployment
Application
Application Monitoring
Container Life-cycle Management Service
Container
VM Life-cycle Management Service
Virtual Machine (VM)
Resource Management Service
Container Placement
VM Placement
Consolidation Service
Resource Allocation Service
VM Allocation Service
Container Allocation Service
Power and Energy Consumption Monitoring Service
Power Monitoring Service
Data Center Management Service
Host

Figure 4.3: ***ContainerCloudSim*** simulator architecture.

allowing development of best practices in a containerized cloud context. In this respect, we developed *ContainerCloudSim* which aims to provide support for modeling and simulation of containerized cloud computing environments including:

- Management interfaces for containers, VMs, hosts, and data centers resources including CPU, memory, storage. Particularly, it should provide the fundamental functionalities such as provisioning of VMs to containers, dynamic monitoring of the system state, and controlling the application execution inside the containers.

- Functionalities which enable researchers to plug in and compare new container scheduling and provisioning policies. Container scheduling policies determine how resources are allocated to containers and virtual machines, and can be extended to allow evaluation of new strategies.

- Investigation of energy efficient resource allocation ability of provisioning algorithms. The simulation environment should provide basic models and entities that can be utilized to evaluate the energy aware provisioning algorithms. To this end, container migration and consolidation have to be supported.

- Support for simulation scalability, as the number of container in a CaaS environment is much higher than the number of virtual machines in a data center.

4.4 Simulator Architecture

ContainerCloudSim follows the same layered architecture of CloudSim, with necessary modifications to introduce the concept of containers. In the proposed architecture of *ContainerCloudSim* (depicted in Figure 4.3), CaaS consists of containerized cloud data centers, hosts, virtual machines, containers, and applications along with their workloads. For efficient management of CaaS, the architecture benefits from multiple layers:

Workload Management Service: This service takes care of clients' application registration, deployment, scheduling, application level performance, and health monitoring.

Container Life-cycle Management Service: This service is responsible for container life-cycle management. This includes creating containers and registering them in the system, starting, stopping, restarting, and migrating containers from a host to another host, or destroying the container. In addition, this component is responsible for managing the execution of tasks which are running inside the container and monitoring their resource utilization.

VM Life-cycle Management Service: This service is responsible for VM management and consist of VM creation, start, stop, destroy, migration, and resource utilization monitoring.

Resource Management Service: This service manages the process of creating VMs/containers on hosts/VMs that satisfy their resource requirements and other placement constraints such as software environment. It consists of four main services:

- **Container Placement Service:** Containers are allocated to the VMs based on a Container allocation policy defined in this service.
- **VM Placement Service:** VMs are allocated to hosts considering a VM allocation policy that is defined in the VM placement service.
- **Consolidation Service:** This service minimizes resource fragmentation by consolidating containers to the least number of hosts.
- **Container Allocation Service:** This service is equipped with policies that determine how VM resources are allocated (scheduled) to containers.
- **VM Allocation Service:** This service is equipped with policies that determine how hosts' resources are allocated (scheduled) to VMs.

Power and Energy Consumption Monitoring Service: This services is responsible for measuring the power consumption of hosts in the data center and is equipped with the necessary power models.

Data Center Management Service: This services is responsible for managing data center resources, powering on and off the hosts, and monitoring the utilization of resources.

4.5 Design and Implementation

For implementing the aforementioned functionalities, CloudSim Discrete Event simulator Core is used to provide basic discrete event simulation functionalities and modeling of basic cloud computing elements. Since CloudSim entities and components communicate through message passing operations, the core layer is responsible for managing the events and handling interactions between components. The main classes of *ContainerCloudSim* are depicted in Figure 4.4. In this section, we go through the details of these classes. *ContainerCloudSim* implementation is constituted of two main parts *simulated elements* and *simulated services*. The simulated elements include:

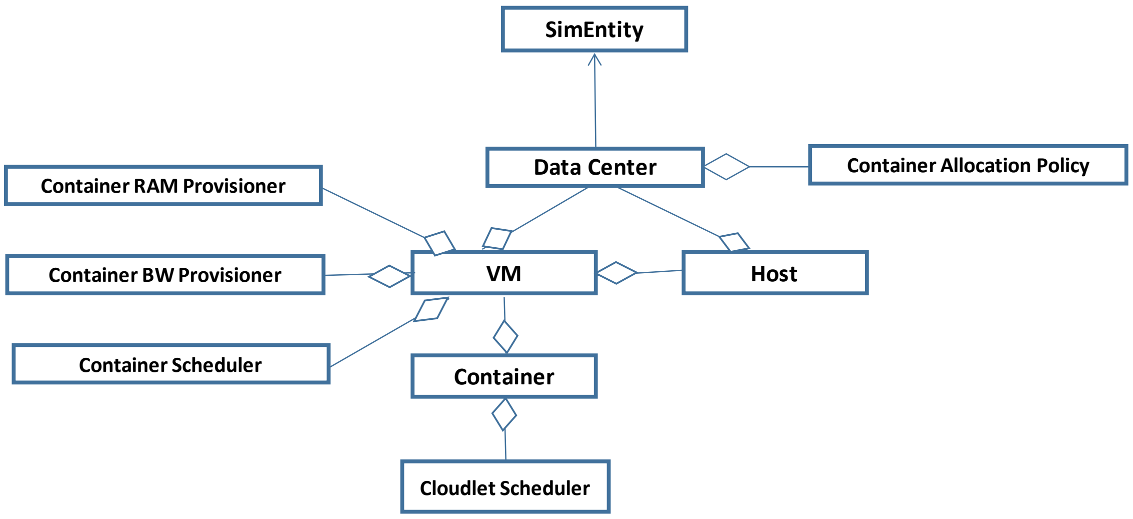


Figure 4.4: ***ContainerCloudSim*** class diagram.

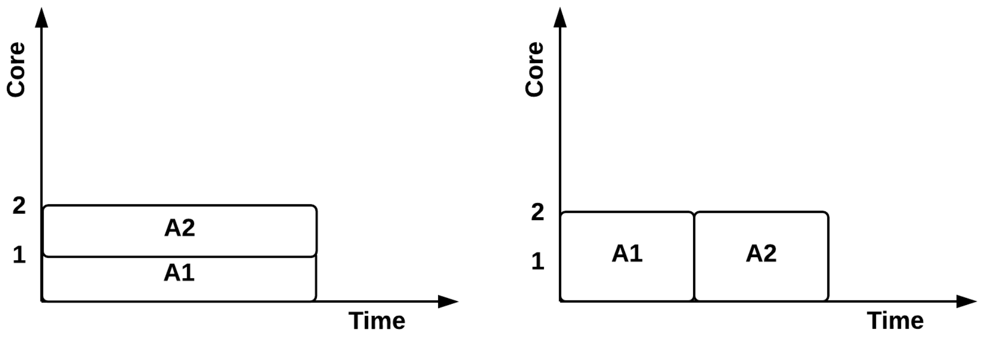
- **Data Center:** The hardware layer of the cloud services is modeled through the Data Center class.
- **Host:** The Host class represents physical computing resources (servers). Their configuration are defined as processing capability that is expressed in MIPS (million instructions per second), memory, and storage.
- **VM:** This class models a Virtual Machine. VMs are managed and hosted by a Host. Attributes of VM are memory, processor, and its storage size.
- **Container:** This class models a Container that is hosted by a VM. Attributes of Containers are accessible memory, processor, and storage size.
- **Cloudlet:** The *Cloudlet* class models applications hosted in a container in cloud data centers. Cloudlet length is defined as Million Instructions (MI) and it has functionalities of its predecessor in CloudSim package including StartTime and status of execution (such as CANCEL, PAUSED, and RESUMED)

In addition, simulated services available in *ContainerCloudSim* are:

- **VM Provisioning:** The VM provisioning policy, which assigns CPU cores from the host to VMs, is considered as a field of the Host class. Similar to CloudSim, the Host component implements the interfaces that provide modeling and simulation of classes that implement CPU cores management. For example, VMs can have dedicated cores assigned to it (pinning of cores to VMs) or can share cores with other VMs in the host.

- **Container Provisioning:** The simulator provides container provisioning at two levels: VM level and container level. At the VM level, the amount of VM's total processing power that is assigned to each container is specified. Whereas at the container level the container can assign a fixed amount of resources to each of the application services that are hosted on it. To enable compatibility with CloudSim, a task unit is considered as a finer abstraction of an application service that is hosted in the container. Time-shared and space-shared provisioning policies are implemented for both levels in the current version of the *ContainerCloudSim* (as depicted in Figure 4.5). In addition, **ContainerRamProvisioner** is an abstract class that represents the provisioning policy utilized for allocating the virtual machine's memory to containers. A container can be hosted on a VM only if the ContainerRamProvisioner component assures that the VM has the needed amount of free memory. If the memory requested by the container is beyond the VM's available free memory, the ContainerRamProvisioner rejects the request. For provisioning the bandwidth, the abstract class named **ContainerBwProvisioner** models the policy for provisioning of bandwidth of the containers. The role of this component is handling network bandwidth allocation to a set of competing containers. This class can be extended to contain new policies to include the requirements of various applications.

  Figure 4.5 illustrates a simple provisioning scenario. In this figure, containers $A1$ and $A2$ are hosted on a host with 2 cores. In the space-shared scenario, only one of the two $A1$ and $A2$ container can run at a given instance of time. Therefore, $A2$ can only be assigned the core when $A1$ finishes its execution. In this scenario, each container requires 2 cores for its execution. However, in the time shared scenario, each container receives a time slice on each processing core and each component receives a variable amount of the processing power during its execution. The available amount of processing power for each container can be estimated through the calculation of the number of active components that are hosted on each VM. The provisioning policy is defined by **ContainerScheduler**, which is an abstract class and is implemented by a VM component. More application-specific processor sharing policies can be implemented by overriding the functionalities of this class.



(a) Space-shared. (b) Time-shared.

Figure 4.5: Space-shared and time-shared provisioning concepts for containers A1 and A2 running on a VM.

- **CloudletScheduler:** The same relationship between container and VM helds between applications (called Cloudlets) and containers. The CloudletScheduler abstract class can be extended to implement different algorithms to identify the share of processing power among Cloudlets that are running in a container. Both types of provisioning policies are included in the *ContainerCloudSim* package, namely time-shared (ContainerCloudletSchedulerTimeShared) and space-shared (ContainerCloudetSchedulerSpaceShared) policies.

- **CloudInformationService:** The CloudInformationService (CIS) class provides resource registration, indexing, and discovering capabilities.

- **ContainerAllocationPolicy:** This abstract class represents a placement policy that is utilized for allocating containers to VMs. The chief functionality of the ContainerAllocationPolicy is to select the available VM in a data center that meets the container's deployment requirements including the container's required memory, storage, and availability. Different placement policies, with different objectives, are created by extending this class.

- **VmAllocationPolicy:** In addition to allocating VMs to hosts, this abstract class implements the *optimizeAllocation* method that defines the consolidation policies in container and VM levels.

- **Workload Management:** Highly variable workloads is one of the main characteristics of cloud applications. In this respect, *ContainerCloudSim* also supports the modeling of dynamic workload patterns of cloud applications in a CaaS environment. We leveraged the existing Utilization Model in CloudSim to determine resource requirements on container-level. The Utilization Model is an abstract class and its getUtilization() method can be overridden by simulator users to obtain various workload patterns. The getUtilization() method input is the simulation time and its output is the percentage of the required computational resource of each Cloudlet.

- **Data Center Power Consumption:** To manage power consumption per host basis, the PowerModel class is included. It can be extended (by overriding the getPower() method) for simulating custom power consumption model of a host. getPower() input parameter is the host's current utilization metric while its output is the power consumption value. By using this capability, *ContainerCloudSim* users are able to design and evaluate energy-conscious provisioning algorithms that demand real-time monitoring of power utilization of cloud system components. The total energy consumption can also be reported at the end of the simulation.

4.5.1 Discrete Event Simulation Dynamics

The simulated processing of task units is managed inside the containers executing the tasks. In this respect, at every simulation step, the task execution progress is updated. Figure 4.6 depicts the sequence diagram of the updating process. At each simulation time step, the method updateVMsProcessing() of the Data Center class is called. updateVMsProcessing() method accepts the current simulation time as its input parameter type. It then calls a method (updateContainersProcessing()) on each host to instruct them to update the processing on each of their VMs. The process is recursively repeated for each VM to update their container processing and for each container to update the application processing. The method at the container level returns the earliest completion time of jobs running on it. At VM level, the smallest completion time among all containers is returned to the host. Finally, at host level the smallest completion time among all VMs is

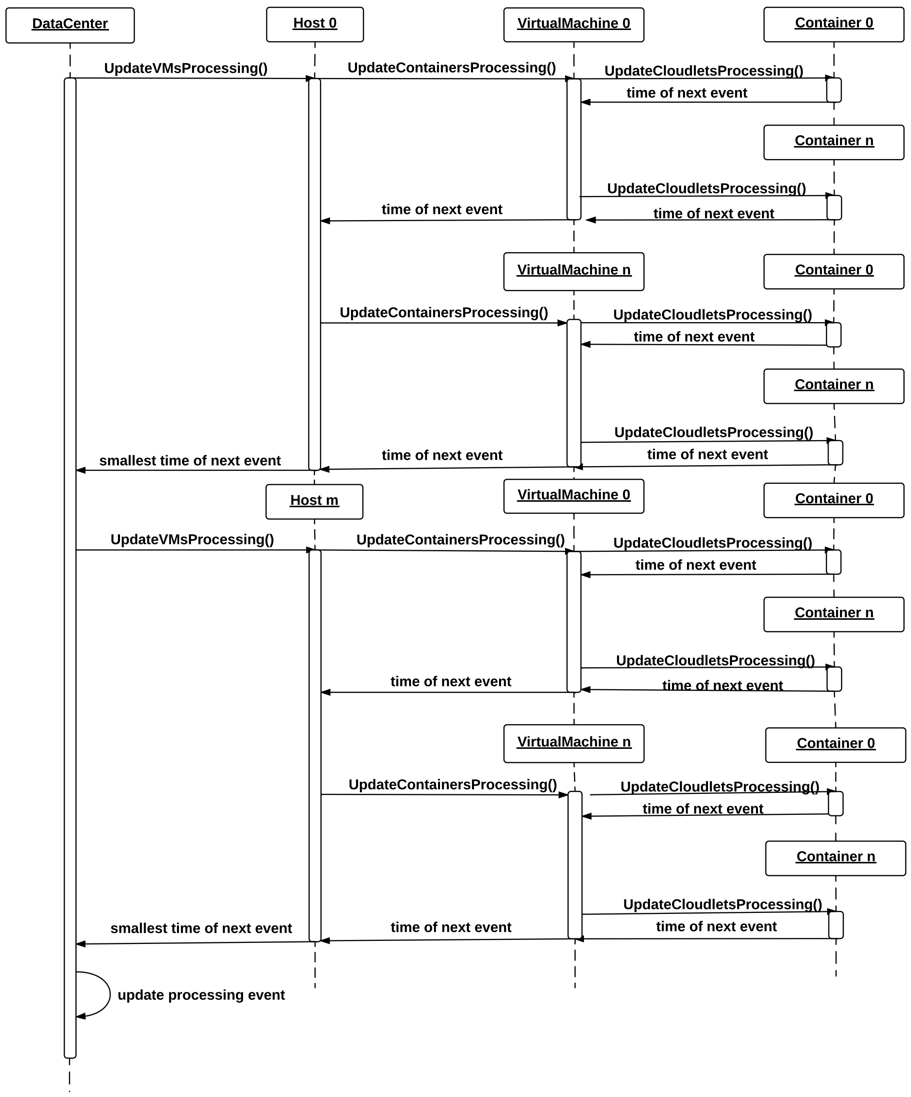


Figure 4.6: Data center internal processing sequence diagram.

returned to Data Center. The earliest time value returned to the Data Center class is used to set the time in which the whole process will be repeated. An event is then scheduled in the simulation core for the calculated time, which dictates the next simulation step, and therefore progresses the simulation.

4.6 Use Cases and Performance Evaluation

To demonstrate the capabilities of *ContainerCloudSim* for evaluating resource management policies, we present three use cases including the container overbooking, container

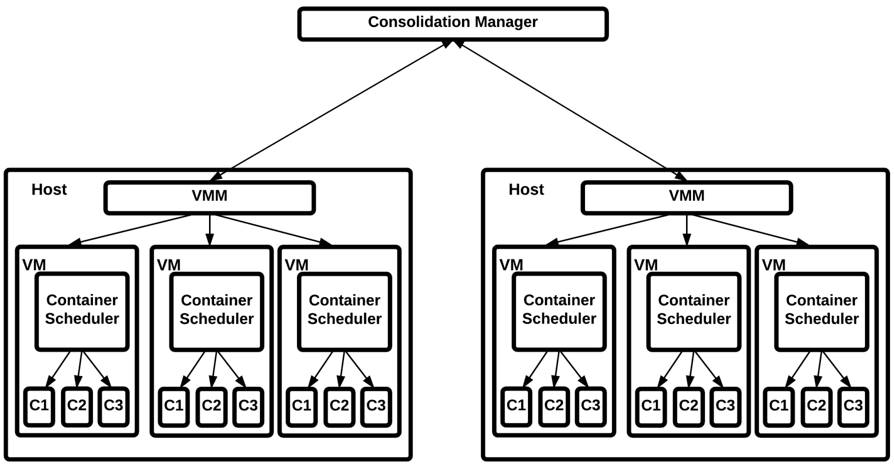


Figure 4.7: A common architecture for the studied use cases: VMM sends the data including the status of the host along with the list of the containers to migrate to the consolidation manager. The consolidation manager decides about the destination of containers and sends requests to provision resources to the selected destination.

consolidation, and container placement. Further, we evaluate the *ContainerCloudSim* in terms of its scalability and container start-up delay modeling.

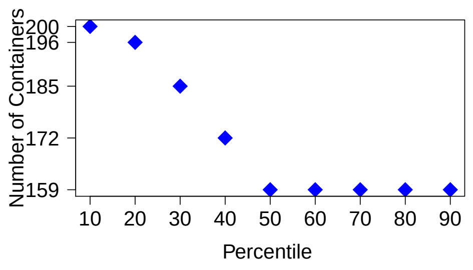
All the use cases leverage the same architecture depicted in Figure 4.7. In this architecture, the VMM deployed on top of physical servers sends the data including the status of the host along with the list of containers that are required to be migrated to the consolidation manager. The consolidation manager, which is deployed on a separate machine, decides about the new placement of containers and sends requests to provision resources to the destination host.

4.6.1 Use Case 1: Container Overbooking

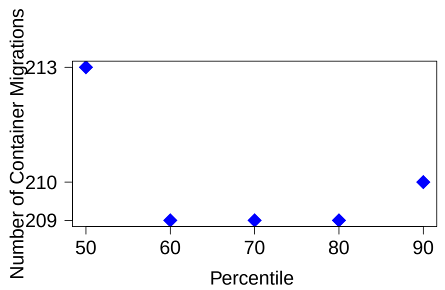
Table 4.1: Configuration of the server, VMs, and containers.

Server Configurations and power models (20 Servers)							
Server type #	CPU [3GHz] (mapped on 37274 MIPS Per core)			Memory (GB)	$P^{\text{idle}}(Watt)$	P^{max} (Watt)	Population
# 1	8 cores			128	93	135	20
Container and VM Types (200 Containers and 20 VMs in total)							
Container Type #	CPU MIPS (1 core)	Memory (GB)	Population	VM Type #	CPU [1.5GHz] (mapped on 18636 MIPS Per core)	Memory(GB)	Population
# 1	4658	128	66	# 1	2 cores	1	5
# 2	9320	256	67	# 2	4 cores	2	5
# 3	18636	512	67	# 3	1 core	4	5
				# 4	8 cores	8	5

Cloud users tend to overestimate the container size they require so that they can avoid the risk of facing less capacity than the actually required by the application. The user's



(a) The number of containers which are successfully allocated to the VMs considering each pre-defined percentiles of the workload.



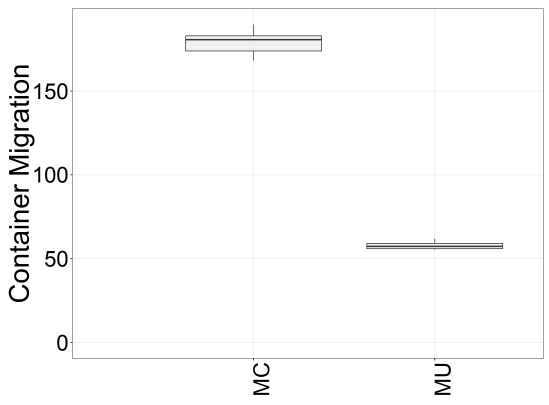
(b) The number of container migrations happened during the experiments.

Figure 4.8: Impact of container's overbooking on the number of successfully allocated containers along with the number of container migrations happened for the experiments with the same number of allocated containers.

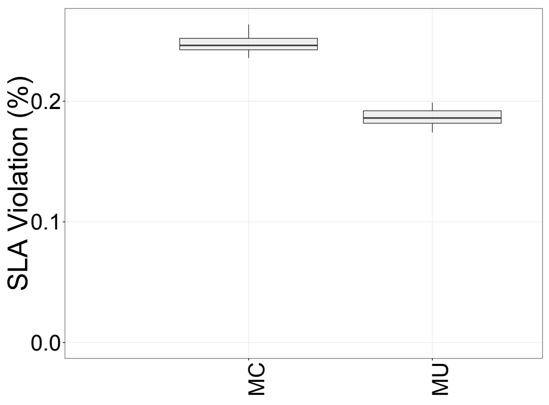
overestimation provides opportunity for the cloud providers to include an overbooking strategy [153] in their admission control system to accept new users based on the anticipated resource utilization rather than the requested amount. Overbooking strategies manage the tradeoff between maximizing resource utilization and minimizing performance degradation and SLA violation. *ContainerCloudSim* is capable of overbooking containers by allocating resources for a specific percentile of the workload.

In this case study, to demonstrate this capacity of the simulator, we designed a couple of experiments to investigate the impact of container overbooking. In the designed experiments, containers are placed on virtual machines according to a pre-defined percentile of their workload, which varied from 10 to 90. The workload traces are derived from PlanetLab [127] and are used as the containers' CPU utilization. These traces contain 10 days of the workload of randomly selected sources from the testbed that were collected between March and April 2011 [15]. In order to eliminate the impact of the container placement algorithm on the results, for all of the studied percentiles, we consider First-Fit as our container placement policy.

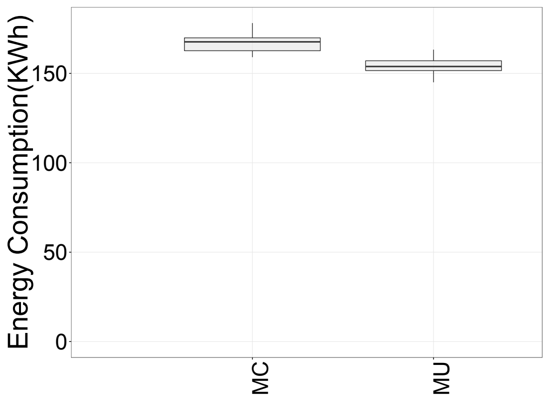
In these experiments, we also utilized the consolidation capability of the simulator. In this respect, the migration process is triggered if the host status is identified as overutilized/underutilized. A simple static threshold-based algorithm is utilized for this purpose. Hosts with less than 70% or more than 80% CPU utilization are considered overloaded or under-loaded respectively. When the migration is triggered because of an overloaded host, the containers with the highest CPU utilization are chosen to migrate.



(a)



(b)



(c)

Figure 4.9: Impact of container selection algorithm on the container migration rate (per 5 minute), SLA violations and the total data center energy consumption.

The simulation setup including the configurations of the servers, containers, and virtual machines are all shown in Table 4.1.

As depicted in Figure 4.8a, the output of the simulation shows that the number of successfully allocated containers decreases as the percentile increases. The higher percentile results in a smaller number of containers accommodated on each VM. The same trend exists when the number of container migrations is considered (Figure 4.8b). The volatility of the workload is the key factor that affects the percentile value. Thus, more volatile workloads would show more difference in the simulation results.

4.6.2 Use Case 2: Container Consolidation

Container consolidation is a promising approach to decrease energy consumption. *ContainerCloudSim* supports this by modeling container migrations aiming at consolidating containers to a smaller number of hosts. In *ContainerCloudSim*, a migration is triggered either because a host is overloaded or under-loaded. To this end, a number of containers should be selected for the migration list in order to rectify the situation. Utilizing *ContainerCloudSim*, researchers are able to study various selection algorithms and investigate the efficiency of their proposed selection policies in terms of the desired metrics including the data center energy consumption, container migration rate, and SLA violations.

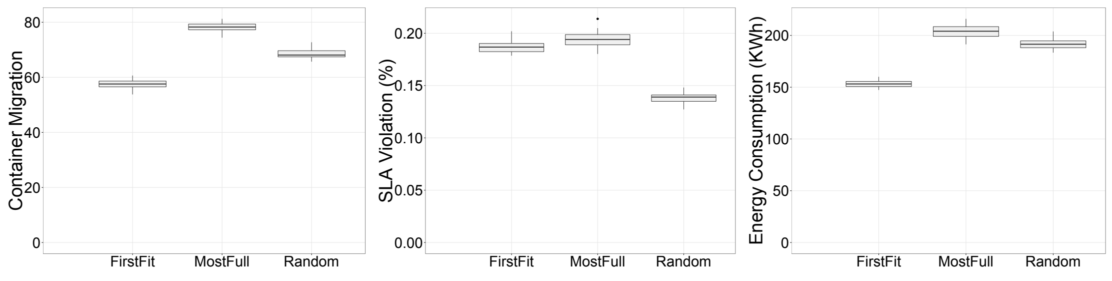
The aim of this case study is utilizing *ContainerCloudSim* to investigate the container selection algorithm effect on the efficiency of the consolidation process of the containers. The same setup, as depicted in Table 4.1, is considered. However, in order to evaluate the algorithms in a larger scale, a larger number of elements are considered in this case study: the number of containers, VMs, and servers set to 4002, 1000, and 700 respectively. Under-load and overload thresholds are fixed as in the previous use case and are 70% and 80% respectively. Containers are placed utilizing the First-Fit algorithm. The destination host is also selected based on the First-Fit policy.

The two algorithms studied for containers selection are “MaxUsage” and “MostCorrelated”. The “MaxUsage” algorithm selects the container that has the biggest CPU utilization while the “MostCorrelated” algorithm chooses the container whose load is the most correlated with the server which is hosting it. Each experiment is repeated 30 times as the workload is assigned randomly to each container. Then, results are compared and depicted in Figure 4.9. The power consumption of the data center at time t ($P_{dc}(t)$) is calculated as $P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t)$, where N_S is the number of servers and $P_i(t)$ corresponds to the power consumption of $server_i$ at time t. CPU utilization is applied for estimating the power consumption of each server as CPU is the dominant component in a server’s power consumption [18]. The linear power model $P_i(t) = P_i^{idle} + (P_i^{max} - P_i^{idle}) * U_{i,t}$ is applied for calculating the servers power consumption, where P_i^{idle} and P_i^{max} are the idle and maximum power utilization of the server respectively and $U_{i,t}$ corresponds to the CPU utilization of server i at time t.

The SLA in this experiment is considered violated if the virtual machine on which the container is hosted does not receive the required amount of CPU that it requested. Therefore, the SLA metric is defined as the fraction of the difference between the requested and the allocated amount of CPU for each VM. The SLA metric is shown in Equation 4.1 [15] in which N_s, N_{vm}, and N_c are the number of servers, VMs and containers respectively. In this equation, $\text{CPU}_{r(vm_{j,i},t_p)}$ and $\text{CPU}_{a(vm_{j,i},t_p)}$ correspond to the requested and the allocated CPU amount to vm_j on server i at time t_p.

$$SLA = \sum_{i=1}^{N_s} \sum_{j=1}^{N_{vm}} \sum_{p=1}^{N_c} \frac{\text{CPU}_r(vm_{j,i}, t_p) - \text{CPU}_a(vm_{j,i}, t_p)}{\text{CPU}_r(vm_{j,i}, t_p)} \tag{4.1}$$

As shown in Figure 4.9, adding the containers with the maximum CPU utilization to the migration list results in less container migrations, energy consumption, and SLA violations and thus should be the preferred policy to be utilized by CaaS providers.



(a) (b) (c)

Figure 4.10: Impact of initial container placement algorithm on the container migration rate (per 5 minute), SLA violations, and data center energy consumption.

4.6.3 Use Case 3: Container Placement Policies

Various mapping scenarios between containers and virtual machines result in different resource utilization patterns. Researchers can utilize *ContainerCloudSim* to study various container to VM mapping algorithms. Therefore, in this case study, we demonstrate how *ContainerCloudSim* is used to investigate the effect of container placement algorithms on the number of container migrations, data center total power consumption, and resulting SLA violations. The same setup as of the **Use Case 2** is applied. The three different placement policies are evaluated: FirstFit, MostFull, and random. As depicted in Figure 4.10, the MostFull placement algorithm, which packs containers on the most full virtual machine in terms of the CPU utilization, results in a higher container migration rate. Consequently, the aforementioned algorithm results in higher violations and energy consumption. In contrast, FirstFit results in less number of migrations and energy consumption, and thus should be the preferred policy to be utilized if the goal of the provider is to reduce energy consumption.

4.6.4 Container and VM Start Up Delays

An important operation in cloud computing environments is instantiation of virtual machines. This time is non-negligible and can impact performance of applications running on clouds. Virtual machine start-up delay of virtual machines was previously studied by Mao et. al [110]. Based on this study, the current version of the simulator includes a static delay of 100 seconds for every virtual machine.

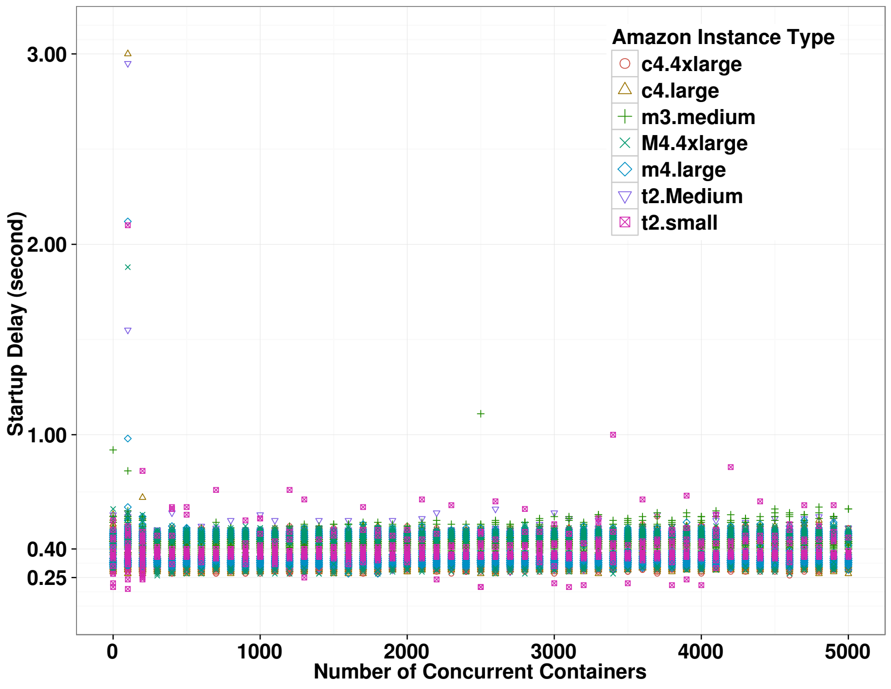


Figure 4.11: The container start up delay for running 1 to 5000 concurrent containers in each of the studied Amazon EC2 instances.

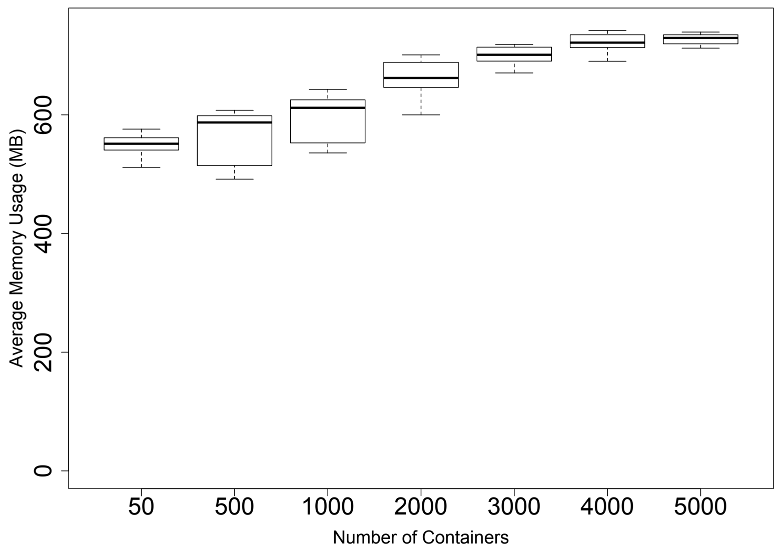
Containers startup delay is also important, since live migration of containers is not applicable in real world scenarios. Therefore, the container migration is performed through shutting down the container on the source host as soon as the same container is started on the destination host. Likewise VMs, container startup delay can be set as a constant (ConstantsEx.Container_STARTTUP_DELAY) in the current version of *ContainerCloudSim*.

The Docker containers startup delay has been recently studied for running one to 100 Ubuntu containers on top of one of the Amazon EC2 instances (c4.4xlarge)[2]. Each container runs the server *Uptime* command. This command is used for identifying how long a system has been running. The storage backend devicemapper is utilized along with an Ubuntu Linux image. In order to have a better understanding of the startup delay, we also followed the same setup. However, we increased the number of containers simultaneously executed from 100 to 5000 (adding one container at a time). The experiment is conducted for other selected Amazon EC2 instance types. As illustrated in Figure 4.11, the start up delays varies between 0.2 to 0.5 seconds for most of the cases. For the current version of the simulator, we utilized the average container startup delay for all studied instance types, which is equal to 0.4 seconds.

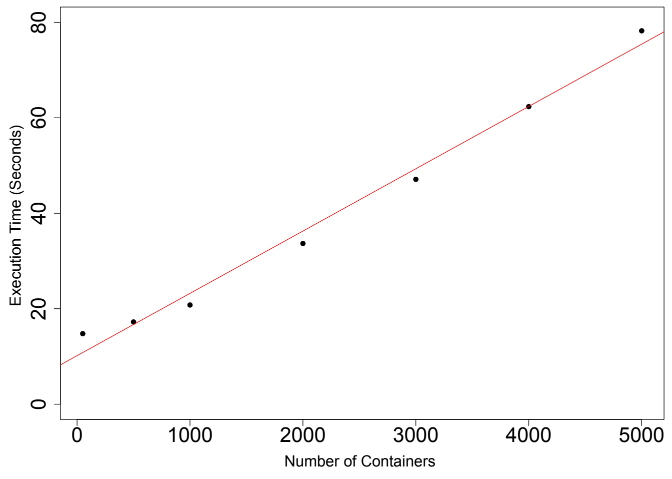
4.6.5 Simulation Scalability

As a containerized cloud simulator, *ContainerCloudSim* scales with minimal memory overhead and execution time. In order to investigate the scalability of the developed simulator, we ran the same experiment setup as Case 2 considering the MostFull algorithm as the container initial placement policy. The same experiment is repeated for various number of containers ranging from 50 to 5000. In order to have a fair comparison, the number of available hosts and VMs are considered constant and equal to 700 and 1000 respectively. For each experiment, the execution time of the simulation, which is defined as the time that it takes for the simulation to finish, and the memory utilization of the Java program (simulation) are depicted in Figure 4.12. The experiment results show (Figure 4.12a) that the memory overhead for running 5000 containers is less than 200MB on average. Considering the execution time, as it is depicted in Figure 4.12b, with every 1000 containers added the simulation time increases by 13 seconds only. It shows that *ContainerCloudSim* is scalable enough to enable simulation experiments on the scale expected in the context of Container as a Service.

[2]Docker Performance Tests: `http://www.draconyx.net/articles/some-docker-performance-tests.html`



(a)



(b)

Figure 4.12: Impact of increasing the number of containers on the average memory usage and the execution time of the simulator.

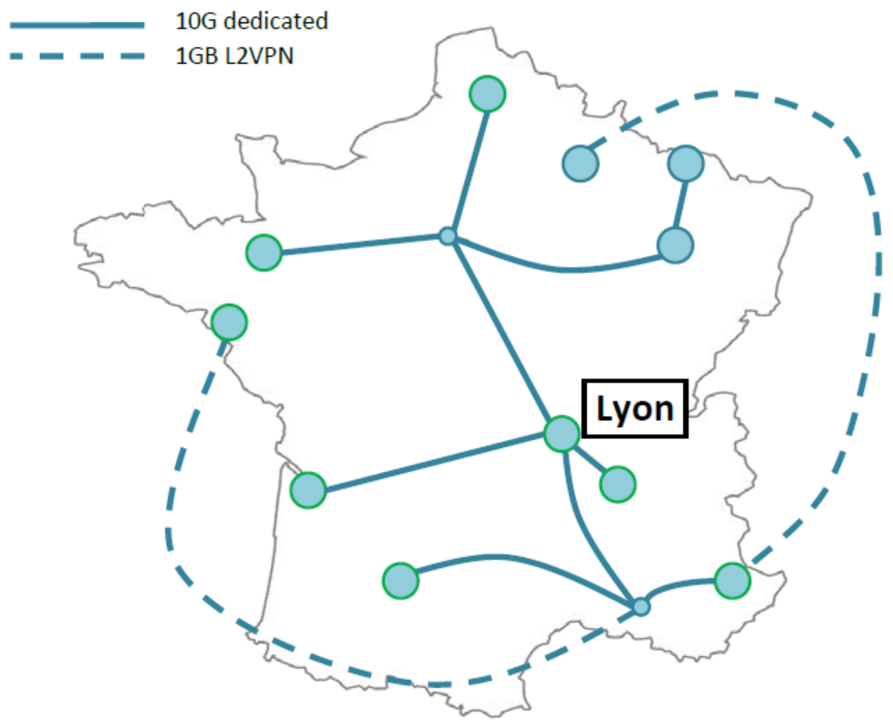


Figure 4.13: Grid5000 infrastructure sites in France. The circles show the sites that are distributed across the country.

4.6.6 Energy consumption overhead of CaaS

Simulators are designed and developed to imitate the real-world behaviour of the systems and processes [9]. As one of the objectives of ContainerCloudSim is enabling researchers to compare the energy efficiency of various algorithms for different kinds of virtualization technologies, it is important to incorporate the impact of these technologies on the server's energy consumption.

ContainerCloudSim is designed in a way that user can plugin various server energy model or the server's power profile for different simulation scenarios. However, apart from the impact that the user's choice of power model would have on the simulation results, one should also consider the overhead of different virtualization technologies on server's energy consumption especially when it comes to their comparison.

In order to investigate the energy consumption overhead of running containers inside virtual machines, we utilized Grid5000 [19] infrastructure in which we are able to monitor the energy consumption of the servers through the provided API. The experiments are carried out using the clusters in Lyon site (See Figure 4.13). As we discussed earlier, CPU is the dominant component in a server's power consumption [18], therefore in our experiments we solely focus on the CPU utilization of our studied server and its impact on its power consumption.

In order to generate CPU intensive workloads, we employed *Stress*[3] package and we invoked the *Stress* command to stress all the available resources (CPU cores) for a specific amount of time (2 minutes for our experiments). Additionally, for limiting the CPU utilization from 10% to 100%, we utilized the *CPU limit* package [4]. For CPU loads the *Stress* command is invoked for 2 minutes stressing all the available CPU cores, while CPU limit is set to the load amount ranging from 10% to 100% with 10% increase in the load for each cycle.

Energy consumption of the available virtualization technologies has been recently studied indicating that containers outperform VMs in terms of both performance and power consumption [53, 117, 165] when running on bare hardware. To the best of our knowledge, energy consumption overhead of running containers in VMs that is the case in our studied cloud service model (CaaS) has not yet been explored. Hence, we studied the energy consumption of a server in three different scenarios. In the first scenario, we considered a bare-metal server with Debian Wheezy[5] as its operating system. While in the second and third scenarios where the hypervisor based virtualization is studied, we deployed the wheezy-x64-xen image[6]that is available on the Grid5000 platform. In the first scenario, the *Stress* command is invoked in the server, while in the second scenario it is used to generate the VM load. In the third scenario, the load is generated in each container while containers are running on top of VMs. The generated load increases every three minutes by 10% ranging from 10% to 100%.

In order to have a fair comparison, we considered the same setup for our virtualized environments in the last two scenarios. We considered 4 VMs with 6 vCPUs, and 128 MB of memory. The operating system of these virtual machines is the same as the host operating system that is SMP Debian Wheezy 3.16.36. We considered Xen version 4.1.4 as the hypervisor. For the container manager, we employed Docker version 1.12.1 and considered one container per virtual machine for the third scenario. CPU of each docker

[3]`http://people.seas.harvard.edu/~apw/stress/`
[4]`http://cpulimit.sourceforge.net/`
[5]`https://www.grid5000.fr/mediawiki/index.php/Wheezy-x64-base-1.5`
[6]`https://www.grid5000.fr/mediawiki/index.php/Wheezy-x64-xen-1.6`

Table 4.2: Power Consumption of Taurus-7 Server

Virtualization type	Average power consumption
Bare Metal	169.68
Xen	190.78
Docker running in Xen VMs	194.88

Table 4.3: Average power consumption (W) of Taurus-7 Server when stressing CPU from 0% to 100% in virtualized environment

Load	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Power	137.16	155	162	172.55	188.31	198.26	201	214	222.3	230.11	233.95

container is not limited so that it can occupy all the available resources of the virtual machine that is hosting the container.

Like [117], we utilized the vCPU pining to dedicate a specific physical CPU to the virtual CPU cores. In the virtualized scenarios, we pin each virtual CPU to one physical core, while one of the VMs share one core of its physical CPU with the Domain-0 of Xen hypervisor. The Domain-0 can specifically run on core 0 of the CPU core that is assigned to it using the *vcpu-set* and *vcpu-pin* commands. We ran the experiments on Taurus-7 machine in Grid5000 lyon site. This machine has 24 cores of CPU, 31GB of RAM, and 557GB of hard disk drive (HDD).

As depicted in Table 4.2, Xen hypervisor increases the average power consumption by 12.43% when compared to the bare metal. As illustrated in this table, running containers on top of virtual machines also increases the average power consumption of the server by 4.1 watts when compared to running *Stress* command on VMs directly. This difference on small scale can be considered almost negligible.

4.6.7 Empirical Evaluation:

In order to investigate the accuracy of the reported power consumptions data in ContainerCloudSim, we repeated the same experiment setup using simulation and studied the reported power consumption. In the simulation, we considered one server, 4 VMs and 4 containers accordingly. In this respect, we utilized average power consumption data of Taurus-7 server when stressing CPU from 0% to 100% in the real setup as depicted in Table 4.3. The simulation results show that when the server load is equal to the loads de-

Table 4.4: Average power consumption (W) reported in Grid5000 versus Container-CloudSim

Load	13%	16%	24%	33%	38%	76%	84%	89%	92%
Grid5000	153.4	160.79	164.6	174.75	186.33	222	225	230	230.55
Container-CloudSim	157.1	159.2	166.2	177.2	185.16	218.98	225.42	229.32	230.87
Error (%)	2.41	0.97	0.94	1.74	0.63	1.35	0.189	0.3	0.14

picted in Table 4.3, reported power consumption exactly matches the real data. However, for the CPU loads between these amounts (e.g. 24%) the power consumption is simulated considering a linear relationship between the cpu utilization and the power consumption for these gaps (e.g. the gap between 10% to 20%). Hence, we studied the reported power consumptions of Taurus-7 when the server load falls between the gaps and observe the difference between the reported and the real power consumption of the server. On average the simulated power consumption shows a negligible difference (around 1%) when compared to the average power consumption of the server we obtained in the real setup (Table 4.4). To further validate simulation results, we developed a system implementing container placement policy and carried out experiments on Grid5000 testbed in France as described below.

Container Placement System:

The system architecture is depicted in Figure 4.14. The *Resource Allocator* component determines the placement of both containers and VMs. It then provides the list of machines, and VMs that are required for hosting docker containers. As shown in Figure 4.14, the output of the *Resource Allocator* is used for deploying the OS images on hosts. When all the images are deployed on the required machines. The *VM Provisioner* component creates VMs in every machines. Later, the *Container Provisioner* component creates containers in every virtual machine that is up and running. In this system, the *Computing Resource Usage Data collector* and *Power Usage Data Collector* components are responsible for collecting the data for the power consumption and resource usages of the machines respectively. The power data is fetched using the available Kwapi API on the Grid5000[7] infrastructure. For logging the utilization of the virtual machines, the *xentop* command is

[7]http://kwapi.readthedocs.io/en/latest/

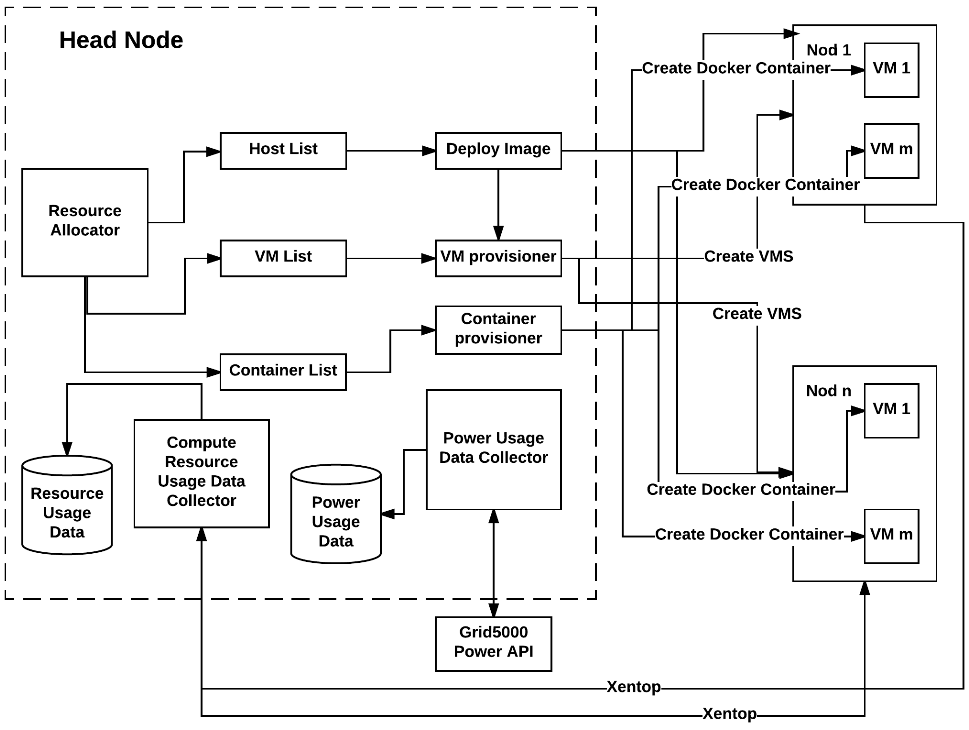


Figure 4.14: The Container Placement System architecture that is employed in the empirical evaluation.

employed to get the average usage of VMs for the last 6 seconds. These two components store the data in their relative databases.

Experiments and Results:

We did this study to demonstrate the accuracy of the reported power usage of our proposed simulator. In the *Resource Allocator* component for both containers and VMs placement, we considered the well-known mostfull algorithm considering the allocated amount of CPU for hosts and VMs. Here, we consider 3 homogeneous servers with 24 cores of CPU, 31GB of RAM, and 557GB of disk. In terms of the virtual machines, we consider 4 VMs per host with 6 cores of CPU, 128 MB of memory, and 10 GB of disk. For containers, we consider 60 homogeneous ones with 1 core of CPU, and 12.8 MB of memory. In the real world setup, if the container's resources are not limited [8], the processes in the

[8]https://docs.docker.com/engine/reference/run/

Table 4.5: Average power consumption (W) reported in Grid5000 versus ContainerCloudSim for container overbooking

Overbooking Percentile	Average Power (W) Grid5000	Average Power (W) ContainerCloudSim	Error (%)	Number of Hosts
20th	448.83	434.45	3.2%	2
80th	625.87	616.18	1.55%	3

container can use as much resource as they need and are available in the VM. The same thing is simulated in ContainerCloudSim, therefore when the containers are assigned to the virtual machines they can occupy all the CPU cores of the VM if available. For cases where more than one container is running in a VM, the VM CPU is divided equally between containers when the VMs resources are fully utilized. For example, when two containers request 100% of the VMs CPU, the virtual machine's operating system assigns 50% of CPU for each container. The workload traces are derived from PlanetLab [127] and are used as the containers' CPU utilization. We utilized *Stress* package to simulate these workloads in the real setup. The container's CPU usage is updated every 3 minutes and the experiment runs for the first 10 CPU load reported by PlanetLab [127].

The experiment is carried out utilizing the Orion cluster of site Lyon of Grid5000 infrastructure. The machines in this cluster has the same architecture and characteristics as Taurus-7 machine that was used for the previous experiment. This experiment is also simulated using ContainerCloudSim and the average power consumption reported by this simulator is compared with the amounts we observed in the real implementation. As we discussed in Section 4.6.1, overbooking containers based on the percentile of the workload affects the number of required virtual machines and consequently the number of required servers as containers are allocated to VMs based on the predefined percentile of the workload. As shown in Section 4.6.1, increasing the percentile increases the number of required hosts and consequently increases the energy consumption. As shown in Table 4.5, for both of the 20th and 80th percentiles the differences between the reported power consumption in the real setup and simulation is less than 5% that can be considered negligible. In addition, both of the approaches (real implementation and simulation) indicate around 30% improvements in terms of energy when considering 20th percentile of the workload for overbooking containers.

4.7 Conclusions

In this chapter, we discussed the modeling and simulation of containerized computing environments as they are currently one of the dominant application deployment models in clouds. We proposed the *ContainerCloudSim* simulator architecture and implemented it as an extension of CloudSim. The simulator architecture is explained in details and we carried out three use cases and demonstrated effectiveness of the *ContainerCloudSim* for evaluating resource management techniques in containerized cloud environment.

The scalability of the simulation is also verified and the approach for modeling container migration is validated in a real environment. Our experiment results demonstrated that *ContainerCloudSim* is capable of supporting simulations on the scale expected in the context of CaaS. *ContainerCloudSim* enables researchers to plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance.

We also verified the accuracy of the power consumption reported by the *ContainerCloudSim*, repeating the experiments in a real setup utilizing the resources of Grid5000 cluster. The results show that the simulator can report the energy consumption with less than 3.2% error. The availability of a container simulation toolkit provides a controllable and repeatable environment for investigation of the container-level resource management algorithms. Hence, in the next chapter, we study the efficiency of a number of container consolidation algorithms and evaluate their performance in terms of energy consumption using *ContainerCloudSim*.

Chapter 5

Efficient Container Consolidation in Cloud Data Centers

One of the major challenges that cloud providers face is minimizing power consumption of their data centers. As we discussed containers are increasingly gaining popularity and going to be major deployment model in cloud environment and specifically in Platform as a Service. This chapter focuses on improving the energy efficiency of servers for this new deployment model by proposing a framework that consolidates containers on virtual machines. We first formally present the container consolidation problem and then we compare a number of algorithms and evaluate their performance against metrics such as energy consumption, Service Level Agreement violations, average container migrations rate, and average number of created virtual machines. We also investigate the virtual machine consolidation efficiency considering the same algorithms applied for the container consolidation problem. We showed that container consolidation is more energy efficient than VM consolidation. Our proposed framework and algorithms can be utilized in any containerized cloud environment including private clouds to minimize energy consumption, or alternatively in a public cloud to minimize the total number of hours the virtual machines leased. The algorithms are evaluated through simulation using our implemented simulator which is briefly discussed in Chapter 4 .

5.1 Introduction

CLOUD computing environments offer numerous advantages including cost effectiveness, on-demand scalability, and ease of management . The aforementioned cloud advantages has encouraged service providers to adopt them and offer solutions via cloud models and consequently encourages platform providers to increase the underlying capacity of their data centers to accommodate the increasing demand of new

This chapter is partially derived from: **Sareh Fotuhi Piraghaj**, Amir Vahid Dastjerdi, Rodrigo N.Calheiros, and Rajkumar Buyya, "A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers ," *Proceedings of the 11th IEEE International Conference on Green Computing and Communications (GreenCom 2015)*, Pages: 368 - 375, Sydney, Australia, 2015.

customers. As mentioned in previous chapters, one of the main drawbacks of the growth in capacity of cloud data centers is the need for more energy for powering these large-scale infrastructures.

The Container as a Service (CaaS) cloud model that has been introduced by Google[1] and Amazon Web Services is increasingly gaining popularity and going to be one of the major cloud service models. A recent study [3] shows that VM-Container configurations obtain close to, or even better performance, than native Docker (container) deployments. However, improving energy efficiency in CaaS data centers has not yet been investigated deeply. Therefore, in this chapter, we use *ContainerCloudSim* to model and tackle the power optimization problem in CaaS.

As we mentioned, servers are still the biggest power consumers in a data center [174]. Therefore, in our proposed framework decreasing the number of running servers is considered as our objective, like the previous chapters. However, in this chapter this objective is met through container consolidation. Like any consolidation solution, our framework should be able to tackle the consolidation problem in three stages. Firstly, it needs to identify the situations in which container migration should be triggered. Secondly, it should select a number of containers to migrate in order to resolve the situation. Finally, it should find migration destinations (host/VM) for the selected containers.

The rest of the chapter is organized as follows. Section 5.2 presents the related work. In Section 5.3, the system objective and problem formulation are presented. Section 5.4 briefly discusses the system architecture and along with its components. Later in Section 5.5, the algorithms are presented. Section 5.6 discusses the testbed and the experiment results. Finally Section 5.7 presents the conclusion of the work.

5.2 Related Work

Unlike the extensive research on energy efficiency of computing [130, 163] and network resources [80, 86, 87, 174] for virtualized cloud data centers, only few works investigated the problem of energy efficient container management.

Ghribi [65] studied energy-efficient resource allocation in IaaS-PaaS hybrid cloud

[1]Google CaaS: `https://cloud.google.com/container-engine/`

model considering both hypervisor-based and containerization technology. Although their proposed algorithms can be applied for both VM and container allocation, the effectiveness of these policies has not been compared for these two virtualization technology. Spicuglia et al. [147] proposed OptiCA, which simplifies the deployment of big data applications in CaaS. The aim of the proposed approach is to achieve the desired performance for any given power and core capacities constraints. OptiCA focuses on effective resource sharing across containers under resource constraints, while we focus on container consolidation to reduce energy consumption.

Dong et al. [45] proposed a greedy container placement scheme, the most efficient server first or MESF, that allocates containers to the most energy efficient machines first. Simulation results, using an actual set of Google cluster traces for modeling as task input and machine set, show that the proposed MESF scheme can significantly improve the energy consumption as compared to the Least Allocated Server First (LASF) and random scheduling schemes.

Yaqub et al. [168] highlighted the differences between deployment models of IaaS and PaaS. They noted that the deployment model in PaaS is based on OS-level containers that host a variety of software services. As a result of unpredictable workloads of software services and variable number of containers that are provisioned and de-provisioned, PaaS data centers usually experience under-utilization of the underlying infrastructure. Therefore, the main contribution of their research is modeling the service consolidation problem as a multi-dimensional bin-packing and applying metaheuristics including Tabu Search, Simulated Annealing, and Late Acceptance to solve the problem. We also modeled the container allocation problem; however, our solution focuses on application of correlation analysis and light-weight heuristics rather than on metaheuristics.

In Chapter 3, we considered CaaS cloud service model and presented a technique for finding efficient virtual machine sizes for hosting containers. To investigate the energy efficiency of our VM sizing technique, we considered baseline scenarios in which virtual machine sizes are fixed. Our approach outperforms the baseline scenarios by almost 7.55% in terms of the data center energy consumption. Apart from the energy perspective, our approach results in less VM instantiations.

Table 5.1: Description of symbols used in Section 5.3.

Symbol	Description
$P_{dc}(t)$	Power Consumption of the data center at time t
$P_i(t)$	Power Consumption of Server i at time t
N_s	Number of servers
P_i^{idle}	Idle power Consumption of Server i
P_i^{max}	Maximum power Consumption of Server i
$U_{i,t}$	CPU Utilization percentage of Server i at time t
N_{vm}	Number of vms
N_{c}	Number of containers
$U_{c_{(k,j,i)}}(t)$	CPU utilization of container k on (VM j, Server i) at time t
N_{v}	Number of SLA Violations
t_p	The time t at which the violation p happened
vm_{ji}	VM j on server i
$\text{CPU}_r(vm_{ji}, t_p)$	CPU amount requested by VM j on server i at time t_p
$\text{CPU}_a(vm_{ji}, t_p)$	CPU amount allocated to VM j at time t_p
$S_{(i,\text{r})}$	Server i Capacity for resource r
$U_{vm_{j,i}(t)}$	CPU Utilization of VM j on Server i at time t
$vm_{(j,i,r)}$	The capacity of resource r of VM j on server i
$c_{(k,j,i,r)}$	The resource r capacity of container k on (VM j, server i)

5.3 System Objective and Problem Formulation

In this section, we briefly discuss the objective of our proposed system, which is minimizing the data center overall energy consumption while meeting the Service Level Agreement (SLA). Firstly, we discuss the power model utilized for estimation of the data center energy consumption and the SLA metric used for comparison of consolidation algorithms. Symbols used in this section are defined in Table 5.1.

5.3.1 Data Center Power Model

The power consumption of the data center at time t is calculated as follows:

$$P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t) \tag{5.1}$$

For estimation of power consumption of servers, we consider the power utilization of the CPU because this is the component that presents the largest variance in power consumption in regards to its utilization rate [18]. Therefore, for each server i, CPU utilization ($U_{i,t}$) is equal to $\sum_{j=1}^{N_{vm}} \sum_{k=1}^{N_c} U_{c_{(k,j,i)}}(t)$ and the power consumption of the server is estimated through Equation 5.2. When there is no VM on the server, it can be switched off to save power. As our studied workload reports the utilization of the containers every five minutes and the migration window is set for 5 minutes accordingly, the servers would be able to boot and become available to process the workload on time. Study [166] show that a blade server can be started and become available in 79 seconds and consumes only 11 watts while it is in its lowest power state (S4).

$$P_i(t) = \begin{cases} P_i^{idle} + (P_i^{max} - P_i^{idle}) * U_{i,t}\,, & N_{\text{vm}} > 0\,; \\ 0\,, & N_{\text{vm}} = 0; \end{cases} \tag{5.2}$$

The energy efficiency of the consolidation algorithms is evaluated based on the data center energy consumption obtained from Equation 5.1.

5.3.2 SLA Metric

Since in our targeted system we do not have any knowledge of the applications running inside the containers, definition of the SLA metric is not straightforward. In order to simplify the definition of the SLA metric, we defined an overbooking factor for provisioning containers on virtual machines which is defined by the costumer based on the percentile of the application workload at the time the container request is submitted. Hence, SLA is violated only if the virtual machine on which the container is hosted on do not get the required amount of CPU that it requested. In this respect, the SLA metric is defined as the fraction of the difference between the requested and the allocated amount of CPU for each VM (Equation 5.3 [15]).

$$SLA = \sum_{i=1}^{N_{\text{s}}} \sum_{j=1}^{N_{\text{vm}}} \sum_{p=1}^{N_{\text{v}}} \frac{\text{CPU}_r(vm_{j,i}, t_p) - \text{CPU}_a(vm_{j,i}, t_p)}{\text{CPU}_r(vm_{j,i}, t_p)} \tag{5.3}$$

5.3.3 Problem Formulation

In order to minimize the power consumption of a data center with M containers, N VMs and K servers, we formulate the problem as follows:

$$min(P_{dc}(t) = \sum_{i=1}^{N_S} P_i(t)) \tag{5.4}$$

Considering the following constraints:

$$\sum_{j=1}^{N_{\text{vm}}} U_{vm_{j,i}}(t) < S_{(i,r)},\ \forall i \in [1, N_s],\ \forall r \in \{CPU\} \tag{5.5}$$

$$\sum_{j=1}^{N_{\text{vm}}} vm_{(j,i,r)} < S_{(i,r)},\ \forall i \in [1, N_s] \\ ,\ \forall r \in \{BW, Memory, Disk\} \tag{5.6}$$

$$\sum_{k=1}^{N_c} U_{c_{(\text{k,j,i})}}(t) < vm_{(j,i,r)},\ \forall j \in [1, N_{vm}] \\ ,\ \forall i \in [1, N_s],\ \forall r \in \{CPU\} \tag{5.7}$$

$$\sum_{k=1}^{N_c} c_{(k,j,i,r)} < vm_{(j,i,r)},\ \forall j \in [1, N_{vm}],\ \forall i \in [1, N_s] \\ ,\ \forall r \in \{BW, Memory, Disk\} \tag{5.8}$$

Equation 5.5 ensures that CPU utilization of virtual machines on each host does not exceed the host's CPU limit. Equation 5.6 ensures that the amount of required memory, disk, and bandwidth of co-located VMs do not exceed the host's resource limits. Equation 5.7 also ensures that the CPU utilization of co-located containers do not exceed the CPU limit of the VM that is hosting the containers. Finally, Equation 5.8 enforces that accumulated memory, disk, and bandwidth requested by the containers are not bigger than the VM's memory, disk, and bandwidth capacity.

Although optimization toolkits can be employed to find a near-optimal solution for the above-mentioned problem, the computation time and complexity increases exponentially with the number of containers. Therefore, in section 5.5 we evaluate a set of heuristic algorithms that can obtain near-optimal solution with less computational overhead.

5.4 System Model

The proposed model targets a CaaS environment where applications are executed on containers. Users of this service submit requests for provisioning of containers. Containers run inside virtual machines that are hosted on physical servers. Both physical servers and VMs are characterized by their CPU performance, memory, disk, and network bandwidth. Likewise, containers are characterized by the demand of the aforementioned resources. The objective of the system is to consolidate containers on the smallest number of VMs and consequently the smallest number of physical servers. The framework consists of 'Host Status' and 'Consolidation' modules that are shown in Figure 5.1 along with their components.

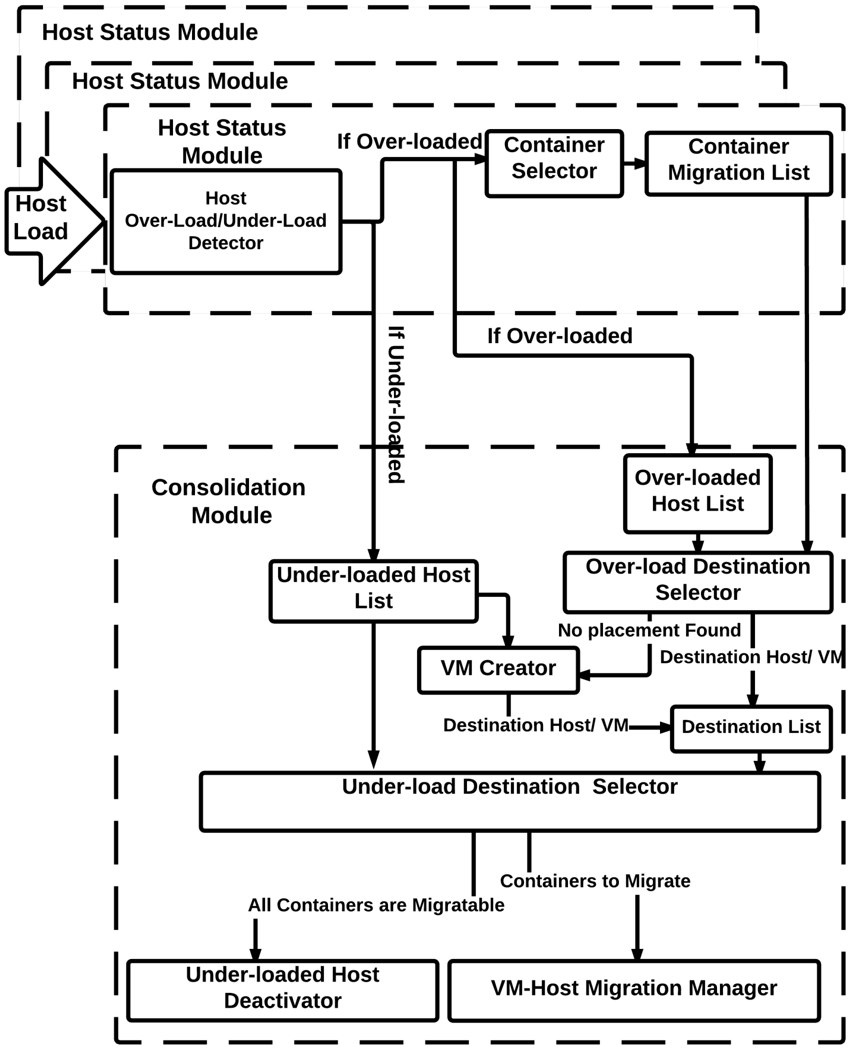


Figure 5.1: System Architecture and Processes.

Algorithm 5: Overview of the Container Selector process.

```
  Input: $serverContainerList(SCL)$
  Output: $SelectedContainersList$
1 while host status is Overloaded do
2     $container \leftarrow$
3     $ContainerSelectionAlgorithm.getContainer(SCL)$
      $SelectedContainersList.add(container)$
4     $SCL.remove(Container)$
```

5.4.1 Host Status Module

The host status module executes on each active hosts in the data center and consists of three main components as follows.

Host Over-load/ Under-Load Detector

The host over-load/under-load detection algorithms, that will be discussed in Section 5.5, are both implemented in this component. The component checks the resource utilization of the host every five minutes. If the host is identified as **under-loaded**, the detector sends the host ID and the IDs of all the containers running on the host to the consolidation module. This is done in an attempt to shut down the under-loaded host if the consolidation module can find new destinations for all the containers. However, if the host status is identified as **over-loaded** the detector sends a request to activate the **Container Selector** component.

Container Selector

The container selection algorithm, which will be discussed in Section 5.5, is implemented in this component. The component is activated whenever the host is identified as **over-loaded** and the container selection process continues until the host status is no longer over-loaded (Algorithm 5).

Algorithm 6: Over-loaded Destination Selection process.

```
   Input: $overLoadedHostList(SCL), CMLs, activeHosts$
   Output: $Destination(hostId, VmId)$
 1 containerList.addAll(CMLs)
 2 $availableHostList(AHL) \leftarrow activeHosts$.removeAll($SCL$)
 3 containerList.sortByCPUUtilization()
 4 foreach $container$ in $containerList$ do
 5     $destination \leftarrow$ HostSelectionAlgorithm.getplacement(AHL,container)
 6     if $destination \neq null$ then
 7         $containerList.remove(container)$
 8         Send $destination$ and $container.getId()$ to Destination List Component
 9     else
10         Send $container$ to VM Creator component
11 Activate the VM Creator component.
```

Container Migration List (CML)

The information about containers selected by the Container Selector are saved in **CML** and are submitted to the consolidation module.

5.4.2 Consolidation Module

The consolidation module is installed on a separate node and can be replicated to avoid single point of failure. This module identifies an appropriate destination for the selected containers to be migrated.

Over-loaded Host List

It stores hosts that are identified as over-loaded by their status.

Over-loaded Destination Selector

This component finds the appropriate destination for the containers in the received CMLs utilizing the host selection algorithm which will be discussed in Section 5.5. The process of this component is shown in Algorithm 6.

Destination List

Data received from the **Overloaded Destination Selector** component, which contains the container ID along with the host and the VM ID of the migration destination, are stored in this list.

VM Creator

This component is responsible for estimating the number of required VMs to be instantiated in the next processing window. The estimation is done based on the number of containers for which the over-load destination selector was unable to assign an appropriate host or VM as the destination. The priority of this component is creating the largest VM possible on under-loaded hosts and assigning the containers to these new VMs. If any container is left, it then chooses a random host from the inactive hosts and creates VMs on this host.

Under-loaded Host List

Hosts that are found to be under-loaded by their status module are stored in this list.

Under-loaded Destination Selector

Considering the under-loaded host list and the destinations decided by the over-load destination selector, this component finds the best destination for containers from the under-loaded hosts (Algorithm 7) using the host selection algorithm discussed in the next section. If this component finds an appropriate destination for all the containers of an under-loaded host, it then sends the destination of the containers together with the destinations decided by the over-loaded destination selector to the VM-Host Migration Manager component. It also sends the host ID to the Under-loaded host deactivator component.

Algorithm 7: Under-loaded Destination Selector process.

```
Input: $destinationList(DL)$, $underloadedHostList(UHL)$,$activeHosts$
Output: $ContainersToMigrateList$
1  $availableHostList(AHL) \leftarrow activeHosts$.removeAll(hosts in $DL$)
2  $ContainersToMigrateList.addAll(DL)$
3  $UHL$.sortByCpuUtilizationInDescendingOrder()
4  foreach $host$ $in$ $UHL$ do
5      if $host.getId()$ $is$ $in$ $DL$ then
6          continue
7      else
8          $AHL$.remove(host)
9          $containerList \leftarrow host.getContainerList()$ foreach $container$ $in$ $containerList$
           do
10             $destination \leftarrow$ HostSelectionAlgorithm.
11                 getplacement(AHL,container)
12             if $destination \neq null$ then
13                 $containerList$.remove($container$)
14                 $tempDestList$.add($destination$)
15         if $containerList.size()$ $is$ $equal$ $to$ $0$ then
16             ContainersToMigrateList.addAll(tempDestList)
17             Send the $host.getID()$ to the Under-loaded host deactivator component.
18         else
19             $AHL$.add(host)
20 Send $ContainersToMigrateList$ to the Migrate to VM-Host Migration Manager
   component.
```

VM-Host Migration Manager

The containers ID together with the selected destinations are all stored by this component, and are used for triggering the migration.

Under-loaded Host Deactivator

It switches off under-loaded hosts that have all their containers migrated.

5.5 Algorithms

In this section, we briefly discuss the algorithms implemented in the components of the 'Host Status' and 'Consolidation' modules of the proposed framework. As we use correlation analysis in the algorithms, we start with a brief description of the Pearson's correlation analysis.

5.5.1 Correlation Analysis

The Pearson's correlation analysis of the container's CPU load X and host (CPU) workload Y performed by the selection algorithms are discussed here. This analysis results in an estimate named "Pearson's correlation coefficient" that quantifies the degree of dependency between two quantities. According to Pearson's analysis, if there are two random variables X and Y with n samples denoted by x_i and y_i, the correlation coefficient is calculated using Equation 5.9 where $\bar{x}$ and $\bar{y}$ denote the sample means of X and Y respectively and r_{xy} varies in the range $[-1, +1]$.

$$r_{xy} = \frac{\sum\limits_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum\limits_{i=1}^{n}(x_i - \bar{x})^2 \sum\limits_{i=1}^{n}(y_i - \bar{y})^2}} \tag{5.9}$$

The more closer the correlation coefficient of X and Y gets to +1, the variables are more likely to have their peak/valley together. In other words, if the container workload is not correlated with the host load, there is less probability of that container causing the host to get over-loaded.

5.5.2 Host Status Monitor Module

We briefly discuss the algorithms that are implemented for each of the module's components including the "Host Overload and under-load Detector" and "Container Selector components".

Overload and Under-load Detection Algorithms

These algorithms are implemented in the *Host Over-load/Under-load Detector* component and are responsible for identifying host status. We consider static thresholds T_{ol} and T_{ul} as the criteria for over-loaded or under-loaded host detection respectively (Equation 5.10).

$$\text{Host Status} = \begin{cases} \text{Overloaded}, & \text{if } U_{(i,t)} > T_{ol} \\ \text{Under-loaded}, & \text{if } U_{(i,t)} < T_{ul} \end{cases} \tag{5.10}$$

Container Selection Algorithms

This algorithm is implemented in the *Container Selector* component and is responsible for selecting a number of containers to migrate from the over-loaded hosts so that the host is no longer over-loaded. The selected container list is saved in the *Container Migration List* and passed to the consolidation module to find a new VM for the containers. We consider two policies as bellow:

- **Maximum Usage (MU) Container Selection Algorithm:** In this policy, the container that has the maximum CPU usage is selected and added to the migration list.
- **Most Correlated (MCor) Container Selection Algorithm:** In this policy, the container that has the most correlated load with the host load is chosen and added to the migration list.

5.5.3 Consolidation Module

The algorithms in this section are implemented in the consolidation module where the new destination is assigned for the CMLs received from the over-loaded hosts and the containers of the under-loaded hosts. The new destination contains the new host ID and the VM ID that the container should be migrated to.

Host Selection Algorithms

The host selection algorithm is implemented in the Overload and Under-load destination Selector components. The output of the algorithm contains the host and VM ID of the migration destination. The following host selection algorithms are studied in this chapter. In all the following selection methods, the virtual machine is chosen using the First-Fit algorithm based on a given percentile of the container's CPU workload.

Algorithm 8: Correlation Threshold Host Selection Algorithm.

```
Input: availableHostList($AHL$), $container$, correlation Threshold ($0 < thr < 1$),
       AlternativeHostSelectionAlgorithm
Output: destination
1  $find \leftarrow false$
2  $containerloadHistory \leftarrow container.getLoadHistory()$
3  if containerloadHistory.size() is less than 5 then
       /* Find a host utilizing a substitute policy. */
4      $destination \leftarrow$ AlternativeHostSelectionAlgorithm.getHost($container$, $AHL$)
5      $find \leftarrow true$
6  while !find do
7      foreach host in $AHL$ do
8          $hostloadHistory \leftarrow host.getLoadHistory()$
9          $cor \leftarrow$Compute the correlation between hostloadHistory and
           containerLoadHistory
10         if $cor < thr$ then
11             if host.allocate(container) then
12                 $destination \leftarrow$host.getId(), host.getVmId(container)
13                 $find \leftarrow true$
14                 break
       /* If no hosts are found then relax the threshold by 0.1
          */
15     $cor \leftarrow cor + 0.1$
16     if $cor > 1$ then
           /* Stop the search, cor can not be bigger than 1. */
17         $destination \leftarrow null$
18         break
```

- **Random Host Selection Algorithm (RHS):** It selects a random host from the available host list *AHL* that can host the container on at least one of its VMs.

- **First Fit Host Selection Algorithm (FFHS):** Chooses the first host of the available host list (*AHL*) that meets the container's resource requirements.

- **Least Full Host Selection Algorithm (LFHS):** The *AHL* list is sorted according to its CPU utilization in descending order. Then, the first host in the sorted *AHL* that meets the resource requirements of the container is selected as the migration destination.

- **Correlation Threshold Host Selection Algorithm (CorHS):** The algorithm first checks

Table 5.2: Server Configurations and power models (700 Servers)

Server type #	Number of CPU Cores, CPU [3GHz] (mapped on 37274 MIPS Per core)	Memory (GB)	p^{idle} (Watt)	p^{max} (Watt)	Population
# 1	4	64	86	117	234
# 2	8	128	93	135	233
# 3	16	256	66	247	233

if the CPU workload history of containers and hosts are adequate for correlation analysis. In a case that the workload history is not available, it simply uses an alternative algorithm such as LHFS and RHS. If the workload history is available, the first host that meets the initial correlation threshold constraint and can accommodate the container on one of its VMs is chosen. If no host is found, the threshold is relaxed by 0.1 until a host is found (Algorithm 8).

5.6 Performance Evaluation

In this section, we discuss the simulation setup along with the experiment results for both container and virtual machine consolidation separately. Further, the results of the most energy efficient host selection algorithm are compared for both container and VM consolidation considering three metrics including energy consumption, average number of migrations and SLA violations.

5.6.1 Simulation Setup

We used *ContainerCloudSim* detailed in Chapter 4 to model a CaaS provider. In our model, we consider 0.4 seconds startup delay for each container[2] and 100 seconds startup delay for VM creation [110]. These startup times are important as they directly affect the SLA metric. SLA metric is affected since migrations would be delayed and hosts would remain in the overload status for a longer amount of time waiting for a container or a VM to become available. In this chapter, we utilized static startup delays for both containers and VMs, however, dynamic startup delays can also be considered for both VMs and containers where more information is available about the application types that are running

[2]Docker Performance Tests: http://sickbits.net/some-docker-performance-tests/

Table 5.3: Configuration of containers and VMs.

Container Types(5000 Containers in total)			
Container Type #	CPU MIPS (1 core)	Memory (MB)	Population
# 1	4658	128	1666
# 2	9320	256	1667
# 3	18636	512	1667

VM Types (1000 VMs in total)			
VM Type #	Number of CPU cores, CPU [1.5GHz] (mapped on 18636 MIPS Per core)	Memory(GB)	Population
# 1	2 cores	1	256
# 2	4 cores	2	256
# 3	1 core	4	256
# 4	8 cores	8	256

inside these components. For example databases might take longer to become available than other kinds of applications such as web servers.

A data center with 700 heterogeneous servers of three different types is simulated (Table 5.2). Characteristics of each server together with VM and container configurations is shown in Table 5.3. Network bandwidth is 1 GB/s, 10 MB/s, and 250 KB/s for servers, VMs, and containers, respectively. Disk bandwidth is 1 TB, 2.5 GB, and 0.1 GB for servers, VMs, and containers, respectively.

In order to evaluate the algorithms considering the aforementioned simulation setup and configurations, we applied the workload traces from PlanetLab [127]. These traces contain 10 days of the workload of randomly selected sources from the testbed that were collected between March and April 2011 [15].

Each container is assigned to one of the workloads containing one day of CPU utilization data which is reported every 5 minutes. In order to consolidate more containers on each virtual machine, a predefined (e.g. 80th) percentile of the workload is considered while packing the containers on the VMs using the First Fit algorithm.

5.6.2 Experiment Results

In this section, we investigate the impact of the algorithms presented in Section 5.5 for both container and virtual machine consolidation in a containerized cloud environment. The main objective is studying the efficiency of the container consolidation when compared to traditional virtual machine consolidation.

Table 5.4: Experiment sets, objectives, and parameters for container consolidation.

Set#	Investigating the Impact of:	Container Selection	UL	OL	Percentile
#1	*OL* Threshold	MU	70%	[80%,90%,100%]	80
#2	*UL* Threshold	MU	[50%,60%,70%]	80%	80
#3	container selection policies	[MU, MCor]	70%	80%	80
#4	overbooking of containers	MU	70%	80%	[20, 40, 80]

Table 5.5: Tukey multiple comparisons of means for energy consumption of the data center for the studied over-load thresholds.

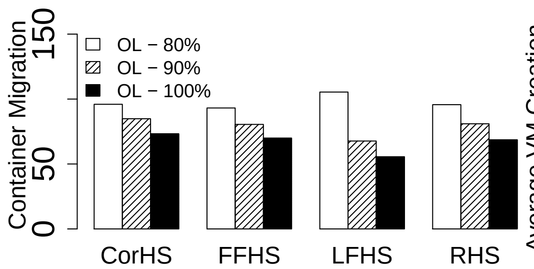
Overload Thresholds	Difference of Means	95% Confidence Interval	P-Value
90% - 80%	-5.93	(-9.40, -2.46)	< 0.001
100% - 80%	-10.42	(-13.89,-6.95)	< 0.001
100% - 90%	-4.49	(-7.93,-1.04)	< 0.001

For both cases, we also studied the effect of the parameters of algorithms on the system performance and data center energy consumption. Four sets of experiments were conducted with different objectives and each experiment on each set is repeated 30 times, the variation in the simulation is resulted from random assignment of workload to each container.

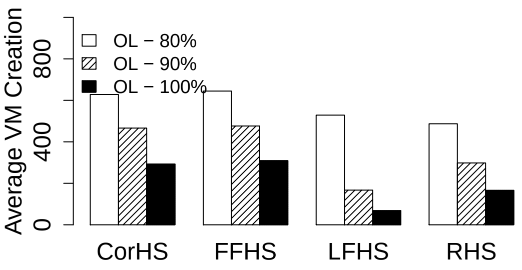
Container Consolidation

The results of the experiments are compared considering four metrics, namely SLA violations (as discussed in Section 5.3), energy consumption, container migrations rate (number of containers migrated in each 5 minutes time slot), and average number of VMs created during the 24 hours simulation period.

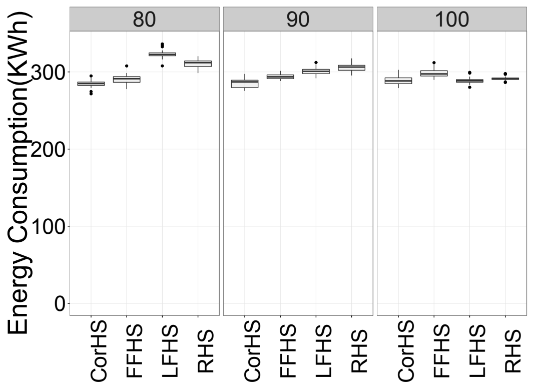
Impact of the Over-load (OL) Threshold: We investigated the effect of the OL threshold in the *Host Over-load/Under-load Detector* component that identifies the host status. As shown in Table 5.4, the *OL* threshold varied from 80% to 100% while the other parameters remained stable . Figure 5.2 shows that, for all the algorithms, increasing *OL* decreases the container migration rate since less hosts would be identified as over-loaded and less number of containers would be chosen to migrate. This decrease in the container migration rate results in less number of VM creations. On the other hand, higher 'OL threshold' increases the probability that the VMs could not get the required resources for the con-



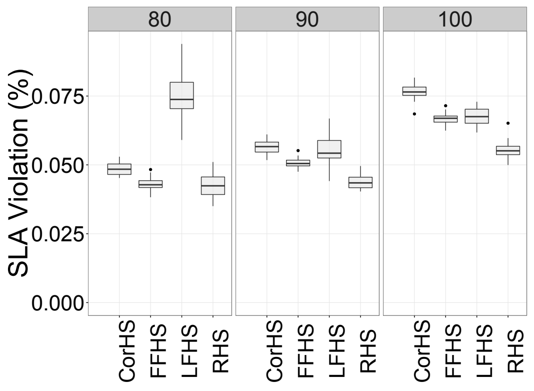
(a) Average container migrations rate per 5 minutes.



(b) Average number of created VMs.



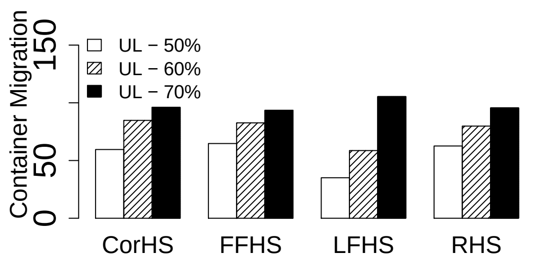
(c) Data center energy consumption.



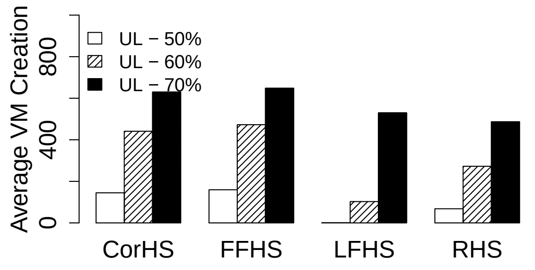
(d) SLA Violations

Figure 5.2: Impact of over-load detection threshold *OL* on container migration rate, created VMs, data center energy consumption, and SLA violations.

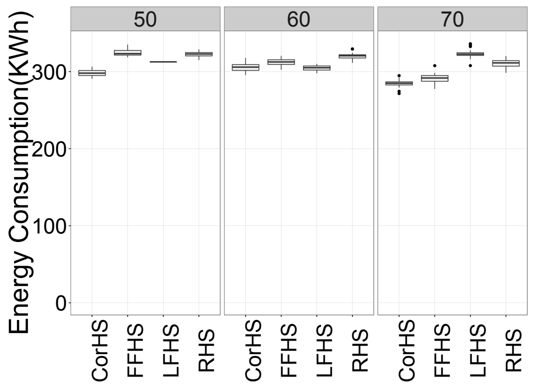
tainers to run and this would increase the SLA violations. The results we obtained from the Anova and Tukey multiple test (See Table 5.5) also verifies that OL threshold has a significant effect with a $P - Value < 0.01$ on the data center energy consumption. For *CorHS* and *FFHS*, 80% is the most efficient threshold in terms of the energy consumption. For *LFHS* and *RFHS* 100% is the best option in terms of the energy efficiency, however as Figure 5.2 depicts average SLA violation considerably raises when compared to 80% and 90% thresholds. *CorHS* with 80% over-load threshold consumes 7.41% less energy on average when compared to other experiments with a reasonable average SLA violations (less than 5%). Performing the Anova and Tukey tests on the 80% OL threshold data verified that the energy consumption improvement by the *CorHS* is significant with a $P - Value < 0.001$ when compared to the other algorithms. When compared to FFHS that is the closes algorithm in terms of the data center energy consumption, *CorHS* shows a 20.22 KWh difference in the mean with a 95% confidence interval between 16.55 and 24.09.



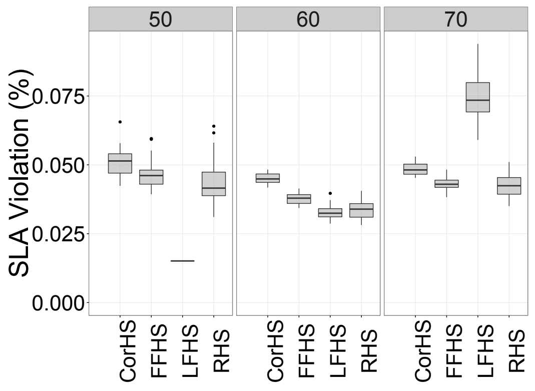
(a) Average container migrations rate per 5 minutes.



(b) Average number of VMs created during the simulation.



(c) Energy Consumption of the data center during the simulation.



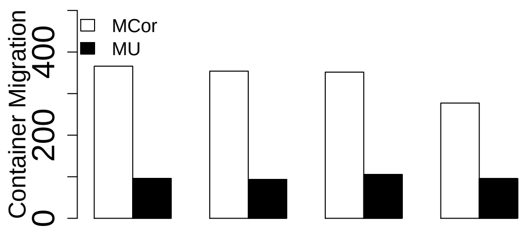
(d) SLA violations.

Figure 5.3: Impact of under-load detection threshold *UL* on container migration rate, created VMs, data center energy consumption, and SLA violations.

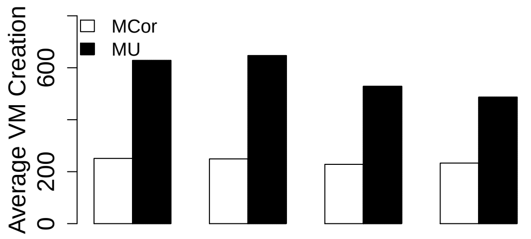
Table 5.6: Tukey multiple comparisons of means for energy consumption of the data center for the studied under-load thresholds.

UL Thresholds	Difference of Means	95% Confidence Interval	P-Value
60% - 50%	-3.64	(-7.27 ,-0.01)	0.049
70% - 50%	-12.04	(-15.67 ,-8.41)	$<$ 0.001
70% - 60%	-8.40	(-12.03 , -4.77)	$<$ 0.001

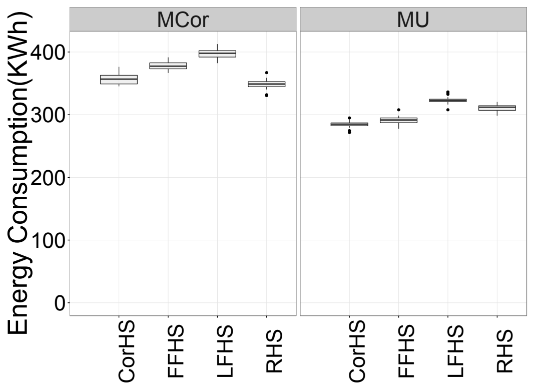
Impact of the Under-load (UL) Threshold: As illustrated in Table 5.4, for this set of experiments we vary the *UL* threshold while keeping the other parameters fixed. As shown in Figure 5.3, decreasing the under-load threshold increases the number of container migrations since more hosts would be identified as under-loaded as the threshold increases. Higher container migration rate results in more VMs to be created to host the migrating containers. This means more SLA violations since the container needs to wait for the VM to start-up. 70% is the most energy efficient threshold for all the algorithms except for *LFHS* because of the bigger gap between the number of the VMs created in 70% under-load threshold and the other two thresholds (Figure 5.3c). Table 5.6 also indicates



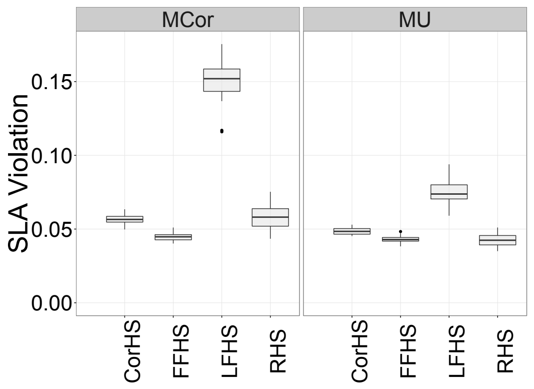
(a) Average container migrations rate per 5 minutes.



(b) Average number of VMs created during the simulation.



(c) Energy Consumption of the data center.



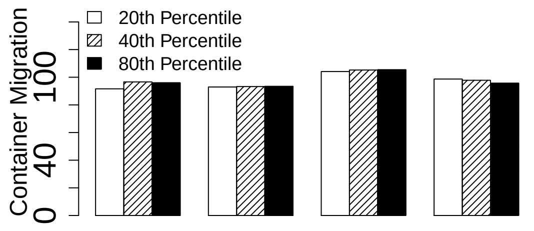
(d) SLA violations.

Figure 5.4: Impact of container selection algorithm on container migration rate, created VMs, data center energy consumption, and SLA violations.

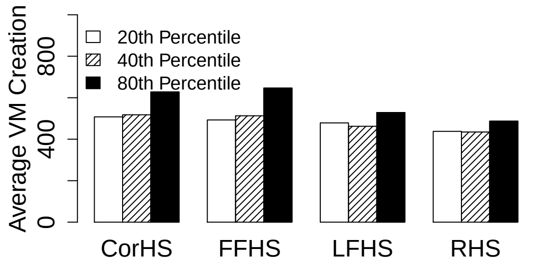
Table 5.7: Tukey multiple comparisons of means for energy consumption of the data center for the studied host selection algorithms considering the MCor container selection algorithm.

Host Selection Algorithms	Difference of Means	95% CI	P-Value
FFHS-CorHS	6.1	(2.40 , 9.8)	< 0.001
LFHS-CorHS	38.37	(34.67 , 42.07)	< 0.001
RHS-CorHS	26.08	(22.377329 , 29.774781)	< 0.001
LFHS-FFHS	32.27	(28.572744 35.970196)	< 0.001
RHS-FFHS	19.98	(16.28 , 23.68)	< 0.001
RHS-LFHS	-12.3	(-16 , -8.6)	< 0.001

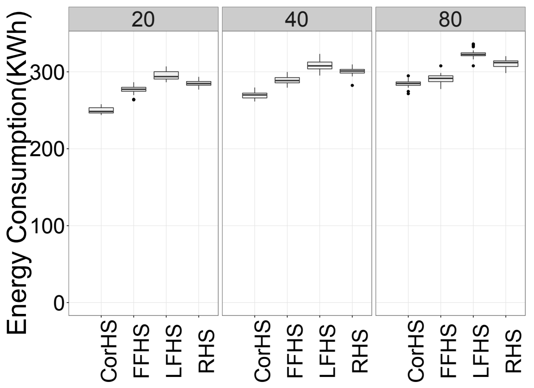
that UL threshold can significantly affect the total energy consumption of the data center, however the difference between the energy consumption for the 50% and 60% thresholds is not significant. *CorHS* with 70% under-loaded threshold, outperforms the other algorithms by 7.46% on average considering energy consumption with less than 5% SLA violations (Figure 5.3). This difference is significant as verified by the Anova and Tukey multiple comparison test with the $P - Value < 0.001$ when compared to other three host selection algorithms.



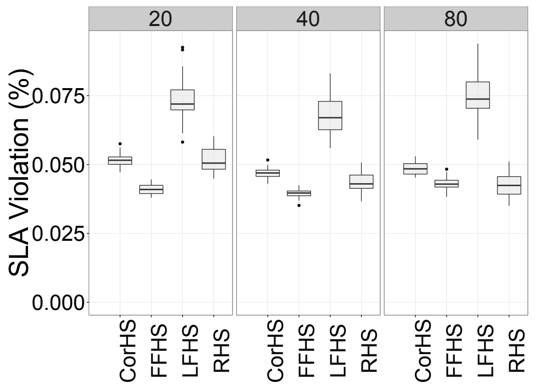
(a) Average container migration rate per 5 minutes.



(b) Average number of created VMs during the simulation.



(c) Energy Consumption of the data center during the simulation.



(d) SLA Violations.

Figure 5.5: Impact of overbooking of containers on migration rate, created VMs, data center energy consumption and SLA violations.

Impact of Container Selection Policies As Figure 5.4 shows, since *MU* chooses the container with the biggest utilization, it requires fewer containers to migrate when the host is over-loaded and consequently results in smaller number of migrations. However, this selection increases the number of VMs required as the migration destination since most of the large containers are selected. Although the delay for starting containers is smaller than for starting VMs, the higher container migration rate in *MCor* results in more SLA violations than *MU*. We carried out T-tests on the energy consumptions reported by all the algorithms, the T-test results show that container selection algorithm significantly affects the amount of energy consumed in the data center with a 95% confidence interval between 63.16 and 72.57 considering the differences between the average (mean) energy consumption of MU (302.37 KWh) and MCor (370.23 KWh). Considering all experiments, *CorHS* is the most energy efficient host selection algorithm with 7.45% less consumption and less than 5% SLA violation. As illustrated in Table 5.7, this difference is significant with a $P - Value < 0.001$.

Table 5.8: Tukey multiple comparisons of means for energy consumption of the data center for the studied overbooking percentile for containers.

Overbooking Percentile	Difference of Means	95% Confidence Interval	P-Value
40-20	15.27	(9.90,20.65)	< 0.001
80-20	25.78	(20.40,31.15)	< 0.001
80-40	10.50	(5.13,15.88)	< 0.001

Table 5.9: Tukey multiple comparisons of means for energy consumption of the data center for the studied host selection algorithms considering the 20th overbooking factor.

Host Selection Algorithms	Difference of Means	95% CI	P-Value
FFHS - CorHS	26.77	(23.39, 30.15)	< 0.05
LFHS - CorHS	45.53	(42.15, 48.91)	< 0.05
RHS - CorHS	34.91	(31.54, 38.29)	< 0.05
LFHS - FFHS	18.76	(15.39, 22.14)	< 0.05
RHS - FFHS	8.14	(4.77, 11.52)	< 0.05
RHS - LFHS	-10.62	(-14, -7.24)	< 0.05

Impact of Container Overbooking: Overbooking is an important factor that affects the efficiency of consolidation algorithms in terms of energy utilization and the SLA violations. Here, containers are allocated to VMs based on the predefined percentile of the application workload running on each container (Table 5.4). The higher percentile results in smaller number of containers accommodated on each VM. Therefore, as Figure 5.5 illustrates 20th percentile results in fewer VMs being created and consequently less energy consumption and more SLA violations. The number of container migrations is the same for most of the algorithms since the variance of the workload is low and migration decisions are based on the host load rather than VM load. As shown in Table 5.8, the overbooking percentile significantly affects the energy consumption of the data center with the $P - Value < 0.001$ for all the studied percentiles. As depicted in Table 5.9, *CorHS* algorithm outperforms the other three algorithms in terms of the energy consumption with a significant difference and a $P - Value < 0.05$ when the overbooking percentile is set to 20 for containers.

VM Consolidation

The same setup and architecture is used for the virtual machine consolidation. However, when a migration is triggered VMs, instead of containers, would be migrated . Therefore, the virtual machines will not be shut down due to the container migrations. Hence, it is

Table 5.10: Experiment sets, objectives, and parameters for VM consolidation.

Set#	Investigating the Impact of:	VM Selection	UL	OL
#1	*OL* Threshold	MU	70%	[80%,90%,100%]
#2	*UL* Threshold	MU	[50%,60%,70%]	80%
#3	VM selection policies	[MU, MCor]	70%	80%
#4	overbooking of containers	MU	70%	80%

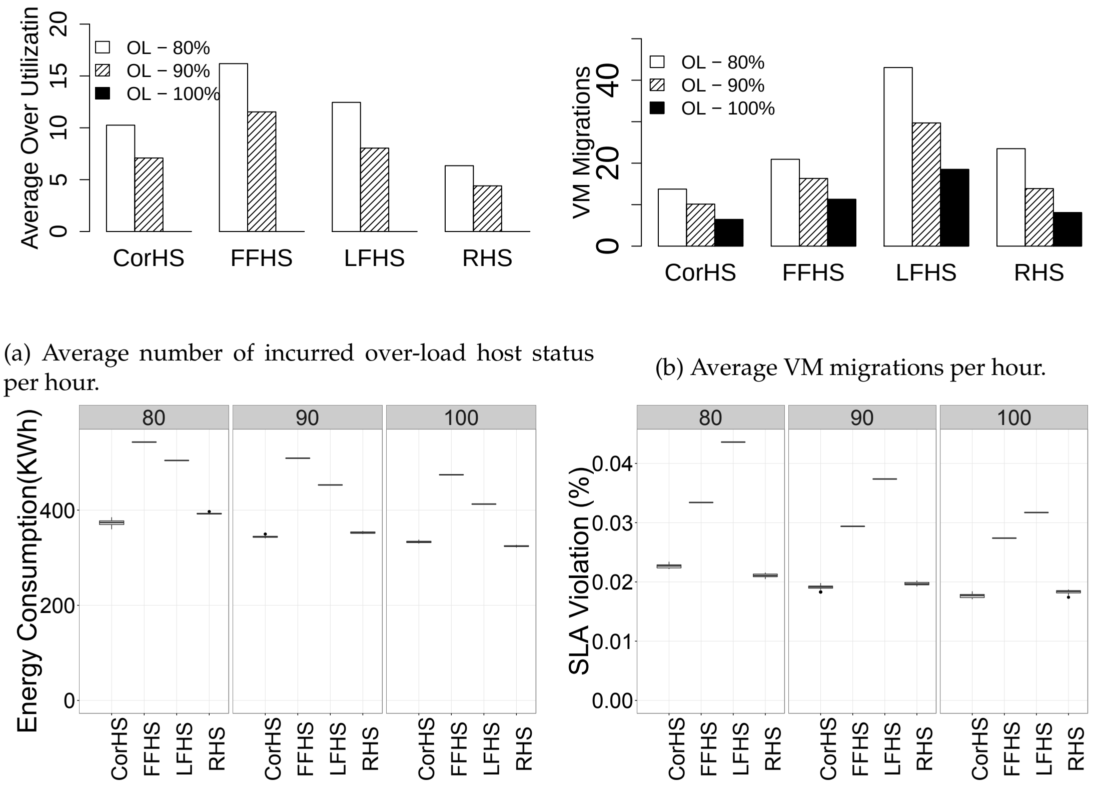
not required to initiate new virtual machines to accommodate the migrated containers.

At the start of the simulation, the containers are placed on the virtual machines utilizing the First-Fit algorithm considering the 80 percentile of its CPU workload. In order to solely study the impact of the correlation of the VM's load with the host's load, random selection (RHS) substitutes the Least-Full algorithm in the correlation aware (CorHS) policy (Algorithm 8). In this respect, if the CPU load of the VM is correlated with all of the running hosts or the data is not enough for drawing conclusion, then a random host is selected as the migration destination. The experiment sets along with their objectives and parameters are summarized in Table 5.10.

Impact of the Over-Load (OL) Threshold: We investigated the effect of the *OL* threshold in the *Host Over-load/Under-load Detector* component that identifies the host status. Figure 5.6 shows that, for all the algorithms, increasing *OL* decreases the number of VM migrations as less hosts would be identified as over-loaded and less number of VMs would be chosen to migrate. This decrease in the average number of VM migrations results in less energy consumption.

The higher *OL* threshold increases the probability that the VMs could not get the required resources. Contrary to the same experiment setup for containers (Section 5.6.2), in this setup of experiments SLA violations increase as the over-load threshold increases. This is resulted from the decreasing pattern of the incurred over-load status while increasing the *OL* threshold. As shown in Table 5.11, OL threshold affects the total energy consumption of the data center significantly considering the P-Values of the Tukey test for the studied thresholds.

For *OL*, equal to 100% the host load never reaches the host capacity as it is depicted in Figure 5.6a. Considering this phenomena, 100% is the most efficient threshold for the studied workload with less than 2% SLA violations (Figure 5.6d). In Table 5.12 for the 100% OL threshold, the difference between the energy consumption of the data center for



(a) Average number of incurred over-load host status per hour.

(b) Average VM migrations per hour.

(c) Data center energy consumption.

(d) SLA Violations

Figure 5.6: Impact of over-load detection threshold *OL* on number of over-load status, average VM migrations (per hour), data center energy consumption, and SLA violations.

Table 5.11: Tukey multiple comparisons of means for energy consumption of the data center for the studied over-load thresholds.

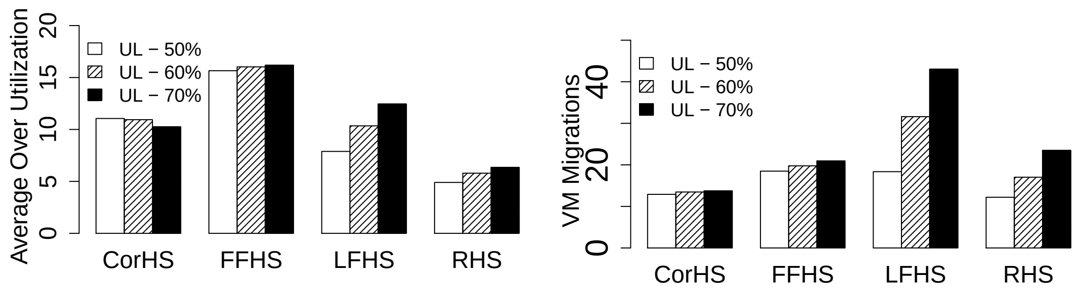
Overload Thresholds	Difference of Means	95% Confidence Interval	P-Value
90% - 80%	-38.68	(-59.36,-18)	< 0.001
100% - 80%	-67.21	(-87.89,-46.53)	< 0.001
100% - 90%	-28.53	(-49.20,-7.85)	< 0.001

the studied host selection algorithms are verified to be significant with the $P - Value <$ 0.01.

Impact of the Under-Load (UL) Threshold: The *UL* threshold that identifies the host status is investigated in this set of experiments and as it is shown in Table 5.10, where the UL threshold varies from 50 to 70 percent. Increasing the underutilization threshold, increases both VM migrations and SLA violations (Figure 5.7). However, it reduces the energy consumption of the data center for most of the algorithms. Although the dif-

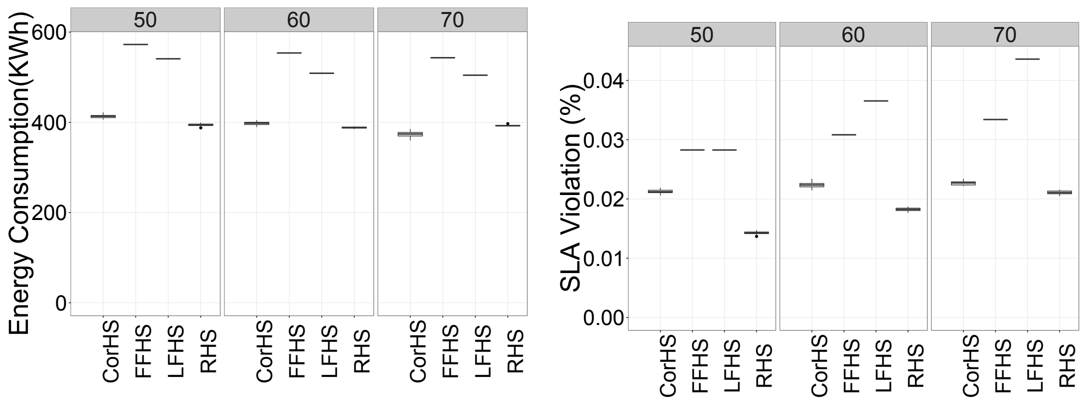
Table 5.12: Tukey multiple comparisons of means for energy consumption of the data center for the studied host selection algorithms considering the 100% OL threshold.

Host Selection Algorithms	Difference of Means	95% CI	P-Value
FFHS-CorHS	169.71	(167.65, 171.76)	< 0.01
LFHS-CorHS	131.06	(129.01,133.11)	< 0.01
RHS-CorHS	19.36	(17.30,21.41)	< 0.01
LFHS-FFHS	-38.64	(-40.67,-36.59)	< 0.01
RHS-FFHS	-150.35	(-152.40,-148.30)	< 0.01
RHS-LFHS	-111.70	(-113.76,-109.65)	< 0.01



(a) Average number of incurred over-load host status per hour.

(b) Average VM migrations per hour.



(c) Energy Consumption of the data center during the simulation.

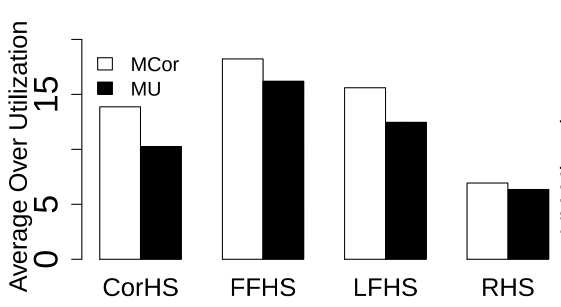
(d) SLA violations.

Figure 5.7: Impact of under-load detection threshold *UL* on number of over-load status, average VM migrations (per hour), data center energy consumption, and SLA violations.

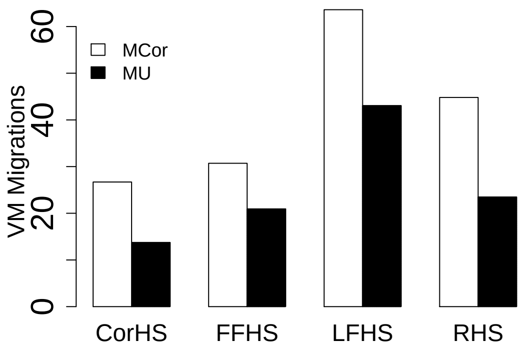
ferences between the energy consumption is not significant for the (50% -60%) and the (60% and 70%) pairs (Table 5.13), the Tukey results show a significant difference between these thresholds in terms of the reported SLA violations with $P - Value < 0.001$. The *UL* threshold increase results in more VM migrations as more hosts would be identified as under-loaded. Due to the raise in the number of VM migrations, more SLA violations

Table 5.13: Tukey multiple comparisons of means for energy consumption of the data center for the studied under-load thresholds.

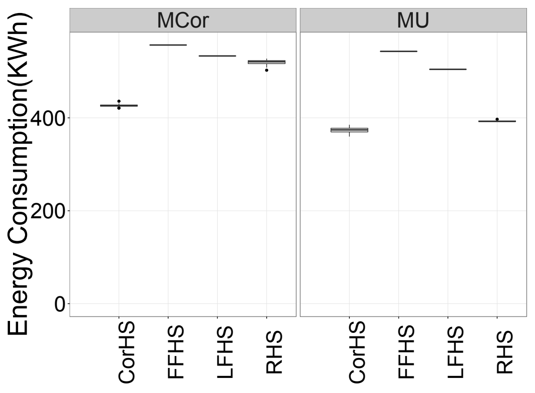
UL Thresholds	Difference of Means	95% Confidence Interval	P-Value
60% - 50%	-18.12	(-40.58,4.33)	0.14
70% - 50%	-26.78	(-49.236,-4.323)	< 0.01
70% - 60%	-8.66	(-31.11, 13.8)	0.67



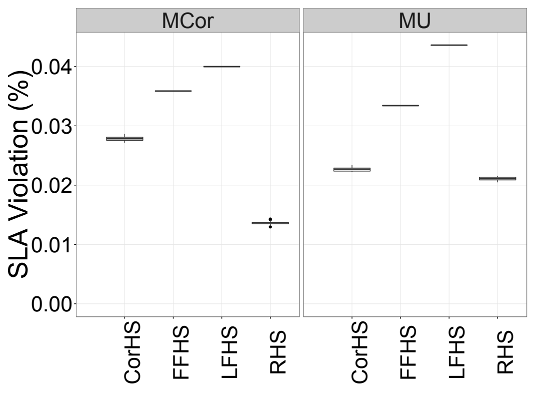
(a) Average container migrations rate per 5 minutes.



(b) Average VM migrations per hour.



(c) Energy Consumption of the data center during the simulation.



(d) SLA violations.

Figure 5.8: Impact of VM selection policies on number of over-load status, average VM migrations (per hour), data center energy consumption, and SLA violations.

are incurred for higher values of under-load thresholds. As depicted in Figure 5.7, 70% is the most energy efficient threshold for all the algorithms. *CorHS*, with 70% under-loaded threshold, outperforms the other algorithms in terms of the energy consumption with less than 3% SLA violations. We also carried out the Anova and the Tukey test for the 70% UL threshold. The test results verify that the difference between the energy consumption of the data center for the *CorHS* algorithm is statistically significant with a $P - Value < 0.01$ when compared to the other studied host selection policies.

Impact of Virtual Machine Selection Policies: Similar to the container selection policies discussed thoroughly in Section 5.5.2, the *MU* policy selects the VM with the biggest CPU utilization while the *MCore* algorithm chooses the VM which has the most correlated CPU load with the host load. Experiment parameters are all shown in Table 5.10.

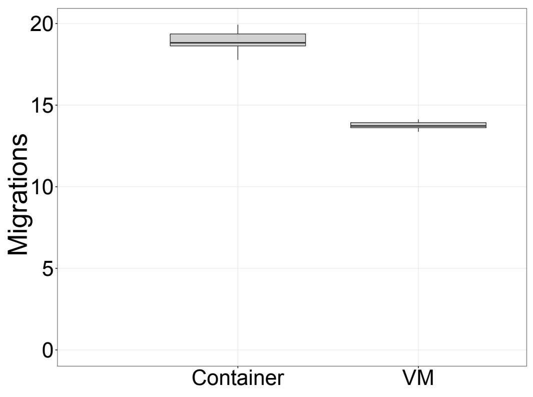
As depicted in Figure 5.8, the *MU* algorithm results in fewer migrations as it selects the largest VMs when the host is over-loaded. The number of over-loaded hosts is higher in *MCore* which shows that selecting a VM with the highest correlation increases the probability of the destination host getting over-loaded. Hence, the *MU* algorithm results in less SLA violations. Considering the energy consumption, *CorHS* algorithm outperforms the other three studied policies with less than 3% SLA violations. We carried out T-tests on the energy consumptions reported by all the algorithms, the T-test results show that container selection algorithm significantly affects the amount of energy consumed in the data center with a 95% confidence interval between 39.68 and 71.26 considering the differences between the average (mean) energy consumption of MU (453.44 KWh) and MCor (508.91 KWh).

5.6.3 Container Consolidation Versus VM Consolidation

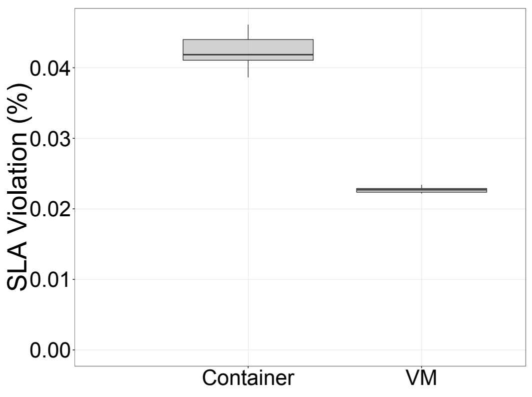
In order to investigate the efficiency of the container consolidation, we compare the container consolidations algorithm with the VM consolidation ones. The same server, virtual machine, and container configurations are considered as they are depicted in Table 5.2, and Table 5.3 respectively. We also considered the same power models which were applied to the previous experiments (Table 5.2).

For comparison purpose, the most energy efficient algorithm for both VM and container consolidation algorithms are selected. For container consolidation, *CorHS* is selected having 70% and 80% as the under-load and over-load thresholds while selecting the most utilized container. From the virtual machine consolidation algorithms considering the aforementioned thresholds, *CorHS* outperforms the other three algorithms in terms of the data center energy consumption.

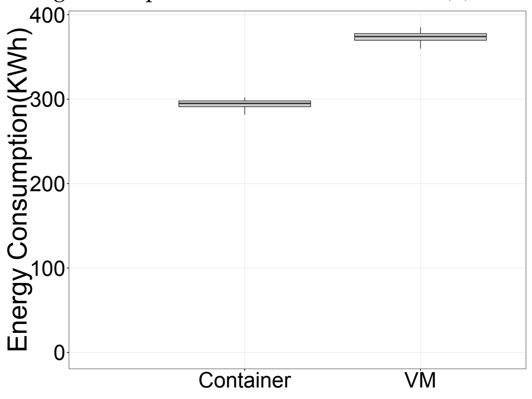
In order to have a fair comparison, for the container consolidation we considered



(a) Average container/VM migrations per hour.



(b) SLA violations.



(c) Energy Consumption of the data center during the simulation.

Figure 5.9: Investigating the efficiency of the Container consolidation versus VM consolidation considering the average number of migrations (per hour), SLA violations, and data center energy consumption.

the Random Host Selection (RHS) algorithm as an alternative if no hosts are found after relaxing the correlation threshold in the *CorHS* Algorithm 8.

As depicted in Figure 5.9, container consolidation is more energy efficient than the virtual machine consolidation with a minimal SLA violation (around 2% more compared to VM consolidation). Therefore, if the extra 2% SLA violations is acceptable for providers, container consolidation can replace VM consolidation to save 15%-20% of energy consumption. We carried out T-tests on the energy consumptions reported in VM consolidation and Container consolidation, the T-test results show consolidating containers significantly affects the amount of energy consumed in the data center with a 95% confidence interval between 76.21 and 82 considering the differences between the average (mean) energy consumption of container consolidation (294.31 KWh) and VM consolidation (373.41 KWh).

5.7 Conclusions

Improving the energy efficiency of cloud data centers is an ongoing challenge that can increase the cloud providers return of investment (ROI) and also decrease the CO2 emissions that are accelerating the global warming phenomenon. Despite the increasing popularity of Container as a Service (CaaS), energy efficiency of resource management algorithms in this service model has not been deeply investigated.

In this chapter, we modeled the CaaS environment and the associated power optimization problem. We proposed a framework to tackle the issue of energy efficiency in the context of CaaS through container consolidation and reduction in the number of active servers. Four sets of simulation experiments were carried out to evaluate the impact on system performance and data center energy consumption of our algorithms for triggering migrations, selecting containers for migration, and selecting destinations. Results show that the correlation-aware placement algorithm (**MCore**) with 70% and 80% as under-load and over-load thresholds, outperforms other placement algorithms when the biggest container is selected to migrate (**MU**).

The same host selection algorithms are studied for the containerized cloud environment when consolidations are happening through VM migrations. We applied the same data center configuration and the containers are overbooked considering the 80 percentile of their CPU workload. The *CorHS* algorithm outperforms the other studied state-of-the-art algorithms in terms of the energy consumption when the parameters are set as the previous container consolidation problem. In order to solely study the effect of the correlation algorithm, the Random Host selection algorithms is considered as the alternative policy of *CorHS*. Results show in a containerized cloud environment where container consolidation is available, migrating containers is more energy efficient than consolidation virtual machines with minimal SLA violations.

Chapter 6

Conclusions and Future Directions

This chapter summarizes the thesis investigation on Energy-Efficient Management of Resources in Enterprise and Container-based Clouds and highlights its main research outcomes. It also discusses open research challenges and future directions in the area.

6.1 Summary

CLOUD, as a utility-oriented computing model, has been facing an increasing adoption rate by various businesses. As stated by RightScale in their 2015 report, "*68% of enterprises run less than a fifth of their application portfolios in the cloud while 55% of them has built a significant portion of their existing application portfolios with cloud-friendly architectures.*" This rapid growth in Cloud computing adoption resulted in the construction of large-scale data centers that require huge amount of electricity to operate. Therefore, improving the energy efficiency of cloud data centers is considered an ongoing challenge that can increase cloud providers' return of investment (ROI) along with reducing CO2 emissions that are accelerating the global warming phenomenon. Hence, in this thesis we tackled the energy efficiency problem in cloud environments.

Docker [114], as a container management engine, in its first year (2015) has been adopted by 13% of surveyed organizations while 35% of the rest were planning to use it[1]. Despite this increasing popularity of containerized data centers, energy efficiency of resource management algorithms in this deployment model has not been deeply investigated in the literature. Hence, in this thesis containerized cloud environment was set as our target cloud service model and we broke down our general goal to decrease energy consumption of data centers as delineated in Chapter 1. We utilized two capabilities of

[1]`http://www.rightscale.com/blog/cloud-industry-insights/cloud-computing-trends-2015-state-cloud-survey`

containerized cloud environments, namely ability to customized VM sizes and to consolidate containers, to achieve our research objectives. In this respect, we proposed techniques to determine VM sizes and investigated algorithms utilized in different stages of the consolidation process. Furthermore, we compared the efficiency of container consolidation with consolidation in terms of data center total energy consumption and incurred SLA violations.

Chapter 2 presented an in-depth review and analysis of prior research in the area of energy efficient resource management in PaaS and CaaS cloud service models. It classified the related work considering three main features, namely the environment where the applications run, the type of workload and application, and the considered SLA metrics. This chapter helped us to identify the open challenges in the area of energy efficiency and define research directions.

In Chapter 3, we tackled the problem of energy consumption resulting from inefficient resource allocation in containerized cloud environments. We presented a brief analysis of the Google cluster data and proposed an architecture for efficient allocation of resources. Later in the same chapter, we proposed a methodology to determine virtual machine configurations considering the extracted task usage patterns. In this regard, tasks were grouped considering their actual resource usage patterns, which are extracted from historic data, through the application of clustering techniques. Along with virtual machines' resource requirements, we also determined the maximum number of tasks that can be accommodated in each VM. Six policies were proposed for estimating task populations residing in each VM type. Later in Chapter 3, we utilized the proposed architecture and investigated the impact of workload characterization on resulting VM configurations. We extended the architecture to incorporate the CaaS cloud service model and modified the VM sizing strategy through limiting the number of tasks that a VM can accommodate according to the available hardware.

Due to the need for testbed environments evaluating the performance of scheduling and allocation policies in containerized cloud data centers, Chapter 4 focused on the modeling and simulation of these computing environments and presented the *ContainerCloudSim* simulator. To the best of our knowledge, this is the first simulator that provides support for modeling and simulation of containerized cloud computing environments.

ContainerCloudSim was implemented as an extension of the cloud simulator CloudSim. This chapter described a number of use cases to demonstrate how researchers can plug in and compare their container scheduling and provisioning policies in terms of energy efficiency and SLA compliance. The scalability of the simulator was verified and the approach for modeling container migration was validated in a real cloud environment.

Utilizing the proposed *ContainerCloudSim*, in Chapter 5 we modeled the CaaS environment and the associated power optimization problem. A framework was proposed to tackle the power consumption issue through container consolidation. We carried out four sets of simulation experiments to evaluate the impact on system performance and data center energy consumption of the algorithms applied for different stages of the consolidation problem. These algorithms contain the policies used for triggering migrations, selecting containers for migration, and selecting migration destinations. The algorithms were evaluated through simulation and the results showed that our proposed correlation-aware placement algorithm outperforms other placement algorithms.

In order to investigate the efficiency of container consolidation in comparison with traditional VM consolidation, we repeated the same set of experiments considering VM migration as the only available option. We studied the same host selection algorithms for the same containerized cloud environment while consolidations were happening through VM migration only. The correlation-aware VM consolidation algorithm again outperformed the other studied state-of-the-art algorithms in terms of energy consumption when the parameters are set as the previous container consolidation problem. The results demonstrated that in a containerized cloud environment where container consolidation is an available option, migrating containers is more energy-efficient than consolidating virtual machines. However, when the number of container migrations is higher than the number of VM migrations due to the small sized containers, container consolidation can result in slightly more SLA violations. In our studied scenario, container migration resulted in 2% more SLA violations than VM consolidation.

6.2 Future Research Directions

Instead of emulating a physical machine as in the case of VMs, containers focus on process isolation in a way that one or more processes can run inside them. As mentioned in Chapter 2, container is not a new concept and has been introduced in the early 80's by the introduction of *chroot* in the Linux kernel [109]. The introduction of process container [1], which is a way to provision and run a standalone process, goes back to 2006. Due to the confusion caused by multiple meanings of the term *containers* in the Linux kernel, containers were then renamed to *Control Groups* (cgroups), which led to the evolution of *Linux Containers* (LXC). LXC, which is known as an operating system level virtualization environment, is built on the Linux kernel.

In addition to OS containers, there exist a new type of containers, called application containers, that are specifically dedicated to one and only one process that runs the residing application. Application containers are considered a new revolution in the cloud era as containers are lightweight, easier to configure and manage, and can decrease start-up times considerably. These containers are the building block of modern PaaS. In this thesis, we considered a couple of challenges in this area including determination of efficient VM sizes for containers and container consolidation algorithms. However, as the containerization technology is evolving rapidly, researchers will face new challenges. Here, we discuss a number of open research challenges that are required to be addressed in order to further advance the area.

6.2.1 Dynamic Virtual Machine Sizing

Containerization technology provides opportunity for execution of multiple containers on one VM, which in turn improves resource utilization at VM level. As it is demonstrated in Chapter 3, VM size has an impact on the total energy consumption of the data center. We tackled the problem of energy efficiency by tailoring virtual machine sizes to the workload. The objective is achieved through characterization of the workload of the Google data center data. Then, one virtual machine size is determined for each cluster of containers. Due to regular provisioning and de-provisioning of containers, required

[1] `http://fabric8.io/gitbook/processContainer.html`

resources for VMs can be unpredictable and the utilization of average resource usage for clustering purposes may lead to resource wastage. Hence, in order to match the configuration of each VM to its residing workload, our work can be extended to support destination and change in VM configurations in runtime. This approach is referred as *dynamic VM sizing* and various techniques such as Statistical Multiplexing can be utilized in this context.

6.2.2 Multi-objective Container Placement Algorithms

Like in the case for virtual machines, effective placement of containers in the data center affects the utilization of compute resources and decreases the probability of containers' future migrations. In this thesis, we considered the problem of initial placement of containers as a multidimensional bin packing problem and applied the well-known First-Fit placement algorithm for locating containers. We assumed that containers are able to run inside any VM. This approach can be extended considering various packing constraints as placement objectives.

In a containerized cloud environment, containers share the kernel of the server or the VM that they are being hosted in. In this respect, during the placement process of containers, the algorithm should take the VM's operating system into consideration as one of the placement objectives. In other words, container running inside a VM with the Windows operating system will not be able to run in a VM with a Linux OS.

The other objective that can be considered during the packing process is the application's workload usage pattern. As we discussed in Chapter 2, each application container specifically executes one process, which is used for running its residing application. Applications are different in terms of their usage patterns as some may consume I/O intensively while others might be compute- or memory-intensive. In this respect, cloud providers can achieve efficient resource utilization by considering the application type during the placement process. For instance, an I/O-intensive web application can be co-located with a compute-intensive application.

Containerization as a virtualization technology enables multi-tenancy in which various workload types can co-exist. Using this technology, providers can overbook the containers and consequently save energy. However, like VMs [122] overbooking containers might affect the performance of containers that are co-located on each VM. Therefore, considering the performance interference phenomenon in the container placement process is another objective that can be considered and this type of placement not only aims at reducing the energy consumption but also aims at decreasing the interference phenomenon.

6.2.3 Network-aware Container Consolidation Algorithms

In addition to required compute resources, network traffic of containers should also be considered to guarantee Quality of Service in a cloud environment and decrease the data center total energy consumption. Hence, virtual machines should satisfy both the aggregated resource utilization of co-located containers and their required bandwidth. This is a complex problem due to its quadratic nature as it is required to consider the communication between each pair of containers. Moreover, applications should be placed in such a way that they can communicate with each other with the least amount of network overhead. In this regard, containers with higher communication rate should be placed on virtual machines that are hosted on one server or on servers with minimum average network path. It is worth mentioning that network-aware placement has been studied for virtual machines and is demonstrated to be effective in terms of data center energy consumption and generated revenue for cloud providers. The consolidation algorithms in Chapter 5 can be extended to consider communications between containers. However, considering the network on container level increases the complexity of the algorithms, as the number of containers is higher than the number of VMs in a cloud data center.

6.2.4 Joint VM and Container Consolidation Algorithms

In Chapter 5, container consolidation is demonstrated to be more energy efficient than VM consolidation, with minimal SLA violation increase (around 2% compared to VM consolidation). One of the reasons of the increase in SLA violations for container consolidation is the higher migration rate of containers. The migration rate is directly related to the size of containers, which are generally smaller than virtual machines. Hence, when a host is identified as overloaded/under-loaded, the number of containers that are required to be migrated to obviate the condition is higher than the number of VMs to be migrated. In this respect, designing algorithms that dynamically select between VM and container migration based on application SLA requirements and the impact on energy consumption can be considered as a future research direction. In addition, future research can also look into approaches that decide about migrating VMs and containers independently. For example, a policy can decide to keep a container of a particular VM in the current host while moving the VM to a different host.

6.2.5 Extending ContainerCloudSim Simulator

The newly introduced Container as a Service cloud model introduces new research directions that are required to be explored in more details. In Chapter 4, we modeled the containerized cloud environments and developed a simulation toolkit. The *ContainerCloudSim* simulator provides a controllable environment in which researchers are able to plug in their proposed container management algorithms. However, there is still room for adding new functionalities and models to this simulator. For instance, *ContainerCloudSim* can be extended to enable visualization of spatio temporal behavior of nodes, VMs and containers in a data center. In order to study network-aware container placement algorithms, the connectivity between containers should be modeled to support various types of applications including Web and MapReduce-like computing environments.

6.2.6 Advanced Container Overbooking Algorithms

Since cloud users tend to overestimate their resource requirements, overbooking of cloud resources is applied to improve utilization of resources and consequently decrease data center energy consumption. However, excessive overbooking results in violation of Service Level Agreement and degradation of application performance.

In Chapter 5, containers are overbooked considering the percentile of CPU workload of containers. Future research can look into other resources such as memory and network, as well as CPU, for overbooking purposes, as ignoring other resources might result into performance degradation of applications. In addition to the percentile method, which assumes a complete knowledge about the workload, future research can employ online policies to estimate resource utilization by container applications and dynamically define the overbooking factor. In addition to future resource utilization of containers, the overbooking policy can also incorporate knowledge about the correlation of container's workloads to define the overbooking factor.

6.2.7 Deploying a Scalable Containerized Testbed

Theoretical analysis of containerized cloud environments is difficult due to the complexity and characteristics of these large scale distributed systems. As discussed in Chapter 4, simulators are powerful tools that can support scalable and repeatable experiments. However, simulators are not able to cover and model all the aspects of a real world experiment. Hence, there is a need for configurable distributed platforms in order to evaluate the performance of scheduling and allocation policies in containerized cloud data centers.

Although there exist a couple of distributed testbeds such as Grid5000 [19] and PlanetLab [127] that researchers can utilize to study containerized environments, they are not specifically tuned for containerization. Therefore, users of these platforms have to set up their own containerized environment while dealing with various technical difficulties such as network and kernel configurations. In this respect, preparing a highly configurable containerized platform for research purpose can expedite research in this area through helping researchers to perform experiments more efficiently without going through the hassle of setting up their own environment. These type of platform can also enhance studies that compare the efficiency of hypervisor-based and OS-based virtualization technologies considering various metrics including energy consumption, network overhead, and response time of applications.

6.3 Final Remarks

Cloud computing adoption is increasing and containerized cloud environments are going to be one of the dominant cloud deployment models. Energy consumption has always been a challenge in cloud data centers as it affects the Return of Investment and data center's carbon footprint. For this newly introduced model, it is really essential to propose new approaches, such as the ones presented in this thesis, which can decrease the energy consumption in cloud data centers. Research such as this conveys the way to a greener environment by significantly reducing the energy it takes to power data centers. We expect contributions of this thesis to energize further innovation and development in containerized cloud environments and to introduce new challenges in this area.

Bibliography

[1] "The network simulator - ns-2," http://www.isi.edu/nsnam/ns/, (Accessed on 11/30/2015).

[2] A. K. Agrawala, J. Mohr, and R. Bryant, "An approach to the workload characterization problem," *Computer*, vol. 9, no. 6, pp. 18–32, 1976.

[3] Q. Ali, "Scaling web 2.0 applications using Docker containers on vSphere 6.0," http://blogs.vmware.com/performance/2015/04/scaling-web-2-0-applications-using-docker-containers-vsphere-6-0.html, 2015, (Accessed on 03/21/2016).

[4] J. Almeida, V. Almeida, D. Ardagna, C. Francalanci, and M. Trubian, "Resource Management in the Autonomic Service-Oriented Architecture," in *Proceedings of the 2006 IEEE International Conference on Autonomic Computing (ICAC 2006)*, June 2006, pp. 84–92.

[5] J. Anselmi, E. Amaldi, and P. Cremonesi, "Service Consolidation with End-to-End Response Time Constraints," in *Proceedings of 34th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2008.)*, September 2008, pp. 345–352.

[6] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[7] B. Arnold, S. A. Baset, P. Dettori, M. Kalantar, I. I. Mohomed, S. J. Nadgowda, M. Sabath, S. R. Seelam, M. Steinder, M. Spreitzer, and A. S. Youssef, "Building the ibm containers cloud service," *IBM Journal of Research and Development*, vol. 60, no. 2-3, pp. 9:1–9:12, March 2016.

[8] M. Assuncao, M. Netto, B. Peterson, L. Renganarayana, J. Rofrano, C. Ward, and C. Young, "CloudAffinity: A framework for matching servers to cloudmates," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium (NOMS 2012)*, April 2012, pp. 213–220.

[9] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation*. Prentice Hall, 2010.

[10] P. Barham *et al.*, "Xen and the art of virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, October 2003, pp. 164–177.

[11] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[12] C. L. Belady and D. Beaty, "Roadmap for datacom cooling," *ASHRAE journal*, vol. 47, no. 12, p. 52, 2005.

[13] A. Beloglazov and R. Buyya, "Energy efficient allocation of virtual machines in cloud data centers," in *Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid 2010)*, May 2010, pp. 577–578.

[14] ——, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *Proceedings of the 8th International Workshop on Middleware for Grids, Clouds and e-Science*, December 2010, pp. 4:1–4:6.

[15] ——, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrent Computing : Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[16] A. Beloglazov, R. Buyya, Y. C. Lee, A. Zomaya *et al.*, "A taxonomy and survey of energy-efficient data centers and cloud computing systems," *Advances in computers*, vol. 82, no. 2, pp. 47–111, 2011.

[17] M. Bichler, T. Setzer, and B. Speitkamp, "Capacity planning for virtualized servers," in *Proceedings of the 16th Annual Workshop on Information Technologies and Systems (WITS 2006)*, 2006.

[18] M. Blackburn, *Five ways to reduce data center server power consumption*. The Green Grid, 2008.

[19] R. Bolze, F. Cappello, E. Caron, M. Daydé, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab *et al.*, "Grid'5000: a large scale and highly reconfigurable experimental grid testbed," *International Journal of High Performance Computing Applications*, vol. 20, no. 4, pp. 481–494, 2006.

[20] J. Bottomley, "Containers and the cloud a match made in heaven," ftp://ftp.kernel.org/pub/linux/kernel/people/jejb/LF-End-User-2013.odp, May 2013, (Accessed on 10/22/2015).

[21] R. Buyya, A. Beloglazov, and J. Abawajy, "Energy-efficient management of data center resources for cloud computing: a vision, architectural elements, and open challenges," in *Proceedings of 16th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2010)*, July 2010, pp. 6–17.

[22] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599–616, 2009.

[23] F. Caglar, S. Shekhar, and A. Gokhale, "A performance Interference-aware virtual machine placement strategy for supporting soft realtime applications in the cloud," *Institute for Software Integrated Systems, Vanderbilt University, Nashville, TN, USA, Tech. Rep. ISIS-13-105*, 2013.

[24] R. Calheiros and R. Buyya, "Energy-efficient scheduling of urgent Bag-of-Tasks applications in clouds through DVFS," in *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*, December 2014, pp. 342–349.

[25] R. N. Calheiros, M. A. Netto, C. A. De Rose, and R. Buyya, "EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of cloud computing applications," *Software: Practice and Experience*, vol. 43, no. 5, pp. 595–612, 2013.

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Software: Practice and Experience*, vol. 41, no. 1, pp. 23–50, 2011.

[27] M. Calzarossa and G. Serazzi, "Workload characterization: A survey," *Proceedings of the IEEE*, vol. 81, no. 8, pp. 1136–1150, 1993.

[28] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova, "Evaluation of the Intel® Core i7 turbo boost feature," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC 2009)*, October 2009, pp. 188–197.

[29] M. Chen, H. Zhang, Y.-Y. Su, X. Wang, G. Jiang, and K. Yoshihira, "Effective VM sizing in virtualized data centers," in *Proceedings of the 2011 IFIP/IEEE International Symposium on Integrated Network Management (IM 2011)*, May 2011, pp. 594–601.

[30] Y. Chen, S. Alspaugh, D. Borthakur, and R. Katz, "Energy efficiency for large-scale MapReduce workloads with significant interactive analysis," in *Proceedings of the 7th ACM European Conference on Computer Systems*, April 2012, pp. 43–56.

[31] Y. Chen, A. S. Ganapathi, R. Griffith, and R. H. Katz, "Analysis and lessons from a publicly available Google cluster trace," University of California, Berkeley, Tech. Rep., 2010.

[32] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of Web Applications in a virtualized cloud computing environment," in *Proceedings of the 9th IEEE International Conference on e-Business Engineering*, October 2009, pp. 281–286.

[33] W. Cirne, F. Brasileiro, J. Sauv, N. Andrade, D. Paranhos, E. Santos-neto, R. Medeiros, and F. C. Gr, "Grid computing for of Bag-of-Tasks applications," in *Proceedings of the 3rd IFIP Conference on E-Commerce, E-Business and EGovernment*, September 2003.

[34] S. P. E. Corporation, "Specpower_ssj2008 results," Available at http://www.spec.org/power_ssj2008/results/. Accessed on 04/06/2015.

[35] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu, "Memory power management via Dynamic Voltage/Frequency Scaling," in *Proceedings of the 8th ACM International Conference on Autonomic Computing*, June 2011, pp. 31–40.

[36] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Magazine Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[37] P. DELFORGE, "Energy efficiency, data centers — NRDC," http://www.nrdc.org/energy/data-center-efficiency-assessment.asp, (Accessed on 02/18/2016).

[38] Q. Deng, D. Meisner, A. Bhattacharjee, T. Wenisch, and R. Bianchini, "CoScale: Coordinating CPU and memory system DVFS in server systems," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2012)*, December 2012, pp. 143–154.

[39] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini, "MemScale: Active low-power modes for main memory," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2011)*, March 2011, pp. 225–238.

[40] G. Dhiman, K. K. Pusukuri, and T. Rosing, "Analysis of Dynamic Voltage Scaling for system level energy management," in *Proceedings of the 2008 Conference on Power Aware Computing and Systems (HotPower 2008)*, 2008, pp. 9–9.

[41] K. Dhyani, S. Gualandi, and P. Cremonesi, "A constraint programming approach for the service consolidation problem," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, A. Lodi, M. Milano, and P. Toth, Eds., 2010, vol. 6140, pp. 97–101.

[42] S. Di and F. Cappello, "GloudSim: Google trace based cloud simulator with virtual machines," *Software: Practice and Experience*, vol. 45, no. 11, pp. 1571–1590, 2015.

[43] S. Di, D. Kondo, and F. Cappello, "Characterizing cloud applications on a Google data center," in *Proceedings of the 42nd International Conference on Parallel Processing (ICPP 2013)*, October 2013, pp. 468–473.

[44] S. Di, D. Kondo, and W. Cirne, "Characterization and comparison of cloud versus grid workloads," in *Proceedings of the 2012 IEEE International Conference on Cluster Computing (CLUSTER 2012)*, September 2012, pp. 230–238.

[45] Z. Dong, W. Zhuang, and R. Rojas-Cessa, "Energy-aware scheduling schemes for cloud data centers on Google trace data," in *Proceedings of the 2014 IEEE Online Conference on Green Communications (OnlineGreencomm 2014)*, November 2014, pp. 1–6.

[46] J. J. Durillo, V. Nae, and R. Prodan, "Multi-objective energy-efficient workflow scheduling using list-based heuristics," *Future Generation Computer Systems*, vol. 36, pp. 221–236, 2014.

[47] J. G. Dy and C. E. Brodley, "Feature selection for unsupervised learning," *The Journal of Machine Learning Research*, vol. 5, pp. 845–889, 2004.

[48] C. Engle, A. Lupher, R. Xin, M. Zaharia, M. J. Franklin, S. Shenker, and I. Stoica, "Shark: Fast data analysis using coarse-grained distributed memory," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, May 2012, pp. 689–692.

[49] K. Ettikyala and Y. R. Devi, "A study on cloud simulation tools," *International Journal of Computer Applications*, vol. 115, no. 14, pp. 18–21, April 2015.

[50] Y. Fang, F. Wang, and J. Ge, "A task scheduling algorithm based on load balancing in cloud computing," in *Web Information Systems and Mining*, F. Wang, Z. Gong, X. Luo, and J. Lei, Eds. Heidelberg, Germany: Springer, 2010, vol. 6318, pp. 271–277.

[51] D. G. Feitelson, *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[52] E. Feller, L. Ramakrishnan, and C. Morin, "Performance and energy efficiency of big data applications in cloud environments: a hadoop case study," *Journal of Parallel and Distributed Computing*, vol. 79, pp. 80–89, 2015.

[53] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," in *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, March 2015, pp. 171–172.

[54] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 37–48, 2012.

[55] D. Ferrari, "Workload characterization and selection in computer performance measurement," *Computer*, vol. 5, no. 4, pp. 18–24, 1972.

[56] F. Fittkau, S. Frey, and W. Hasselbring, "CDOSim: Simulating cloud deployment options for software migration support," in *Proceedings of the IEEE 6th International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2012)*, September 2012, pp. 37–46.

[57] A. Forestiero, C. Mastroianni, M. Meo, G. Papuzzo, and M. Sheikhalishahi, "Hierarchical approach for green workload management in distributed data centers," in *Proceedings of the 20th International European Conference on Parallel and Distributed Computing (Euro-Par 2014) Workshops*, August 2014, pp. 323–334.

[58] B. A. Forouzan, *TCP/IP protocol suite*. McGraw-Hill, Inc., 2002.

[59] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, June 2007.

[60] S. Frey and W. Hasselbring, "Model-based migration of legacy software systems into the cloud: The CloudMIG approach," *Softwaretechnik-Trends*, vol. 30, no. 2, pp. 84–85, 2010.

[61] J. Gantz and D. Reinsel, "Extracting value from chaos," https://www.emc.com/collateral/analyst-reports/idc-extracting-value-from-chaos-ar.pdf, June 2011, (Accessed on 11/02/2015).

[62] S. Garg and R. Buyya, "NetworkCloudSim: Modelling parallel applications in cloud simulations," in *Proceedings of the Fourth IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, December 2011, pp. 105–113.

[63] P. Garraghan, I. S. Moreno, P. Townend, and J. Xu, "An analysis of failure-related energy waste in a large-scale cloud environment," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 2, pp. 166–180, 2014.

[64] P. Garraghan, P. Townend, and J. Xu, "An analysis of the server characteristics and resource utilization in Google cloud," in *Proceedings of the 2013 IEEE International Conference on Cloud Engineering (IC2E 2013)*, March 2013, pp. 124–131.

[65] C. Ghribi, "Energy efficient resource allocation in cloud computing environments," Ph.D. dissertation, Evry, Institut national des télécommunications, 2014.

[66] D. Gmach, J. Rolia, and L. Cherkasova, "Selling t-shirts and time shares in the cloud," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2012)*, May 2012, pp. 539–546.

[67] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Resource pool management: Reactive versus proactive or let's be friends," *Comput. Netw.*, vol. 53, no. 17, pp. 2905–2922, Dec. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2009.08.011

[68] R. P. Goldberg, "Survey of virtual machine research," *Computer*, vol. 7, no. 9, pp. 34–45, 1974.

[69] C. D. Graziano, "A performance analysis of Xen and KVM hypervisors for hosting the xen worlds project," 2011, Graduate Theses and Dissertations, Paper 12215. [Online]. Available: http://lib.dr.iastate.edu/etd/12215

[70] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 1, pp. 68–73, 2008.

[71] S. Greenberg, E. Mills, B. Tschudi, P. Rumsey, and B. Myatt, “Best practices for data centers: Lessons learned from benchmarking 22 data centers,” in *Proceedings of the ACEEE Summer Study on Energy Efficiency in Buildings*, August 2006, pp. 76–87.

[72] D. Greenwood, G. Vitaglione, L. Keller, and M. Calisti, “Service Level Agreement management with adaptive coordination,” in *Proceedings of the 2006 International conference on Networking and Services*, July 2006, pp. 45–45.

[73] G. Gu, R. Perdisci, J. Zhang, and W. Lee, “BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection,” in *Proceedings of the 17th Conference on Security Symposium (SS 2008)*, August 2008, pp. 139–154.

[74] S. K. S. Gupta, A. Banerjee, Z. Abbasi, G. Varsamopoulos, M. Jonas, J. Ferguson, R. R. Gilbert, and T. Mukherjee, “GDCSim: A simulator for green data center design and analysis,” *ACM Transactions on Modeling and Computer Simulation*, vol. 24, no. 1, pp. 3:1–3:27, 2014.

[75] A. F. M. Hani, I. V. Paputungan, and M. F. Hassan, “Renegotiation in Service Level Agreement management for a cloud-based system,” *ACM Computer Survey*, vol. 47, no. 3, pp. 51:1–51:21, 2015.

[76] J. Hartigan and M. Wong, “Algorithm as 136: A K-means clustering algorithm,” *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[77] B. G. Haskell, A. Puri, and A. N. Netravali, *Digital Video: An Introduction to MPEG-2*. Springer Science & Business Media, 1997.

[78] Y. He, Z. Ye, Q. Fu, and S. Elnikety, “Budget-based control for interactive services with adaptive execution,” in *Proceedings of the 9th International Conference on Autonomic Computing*, 2012, pp. 105–114.

[79] R. Hecht-Nielsen, “Theory of the backpropagation neural network,” in *Neural Networks for Perception (Vol. 2)*, H. Wechsler, Ed. Harcourt Brace & Co., 1992, pp. 65–93.

[80] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "ElasticTree: Saving energy in data center networks," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, March 2010, pp. 249–264.

[81] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, April 2011, pp. 295–308.

[82] U. Hoelzle and L. A. Barroso, *The Datacenter As a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 1st ed. Morgan and Claypool Publishers, 2009.

[83] S. Hosseinimotlagh, F. Khunjush, and R. Samadzadeh, "SEATS: smart energy-aware task scheduling in real-time cloud computing," *The Journal of Supercomputing*, vol. 71, no. 1, pp. 45–66, 2015.

[84] W. Hu, A. Hicks, L. Zhang, E. M. Dow, V. Soni, H. Jiang, R. Bull, and J. N. Matthews, "A quantitative study of virtual machine live migration," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, August 2013, p. 11.

[85] Y. Jararweh, Z. Alshara, M. Jarrah, M. Kharbutli, and M. N. Alsaleh, "TeachCloud: a cloud computing educational toolkit," *International Journal of Cloud Computing*, vol. 2, no. 2-3, pp. 237–257, 2013.

[86] J. Jiang, T. Lan, S. Ha, M. Chen, and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM 2012)*, March 2012, pp. 2876–2880.

[87] H. Jin, T. Cheocherngngarn, D. Levy, A. Smith, D. Pan, J. Liu, and N. Pissinou, "Joint Host-Network optimization for energy-efficient data center networking," in *Proceedings of the 27th IEEE International Symposium on Parallel Distributed Processing (IPDPS 2013)*, May 2013, pp. 623–634.

[88] M. Kabir, G. Shoja, and S. Ganti, "VM placement algorithms for hierarchical cloud infrastructure," in *Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science (CloudCom 2014)*, December 2014, pp. 656–659.

[89] A. Kansal, F. Zhao, J. Liu, N. Kothari, and A. A. Bhattacharya, "Virtual machine power metering and provisioning," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC 2010)*, June 2010, pp. 39–50.

[90] J. M. Kaplan, W. Forrest, and N. Kindler, "Revolutionizing data center energy efficiency," Technical Report, available at http://www.mckinsey.com/clientservice/bto/pointofview/pdf/revolutionizing_data_center_efficiency.pdf. Accessed on 5/06/2015, 2008.

[91] R. E. Kass and L. Wasserman, "A reference bayesian test for nested hypotheses and its relationship to the schwarz criterion," *Journal of the American Statistical Association*, vol. 90, no. 431, pp. 928–934, 1995.

[92] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[93] R. T. Kaushik, M. Bhandarkar, and K. Nahrstedt, "Evaluation and analysis of GreenHDFS: A self-adaptive, energy-conserving variant of the hadoop distributed file system," in *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, November 2010, pp. 274–287.

[94] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Proceedings of the 2012 IEEE Network Operations and Management Symposium (NOMS 2012)*, April 2012, pp. 1287–1294.

[95] A. Khosravi, S. K. Garg, and R. Buyya, "Energy and carbon-efficient placement of virtual machines in distributed cloud data centers," in *Proceedings of the 19th International European Conference on Parallel and Distributed Computing (Euro-Par 2013)*, July 2013, pp. 317–328.

[96] J. Kim, S. Elnikety, Y. He, S.-w. Hwang, and S. Ren, "QACO: exploiting partial execution in web servers," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference (CAC 2013)*, August 2013, pp. 12:1–12:10.

[97] J. Kim, M. Ruggiero, D. Atienza, and M. Lederberger, "Correlation-aware virtual machine allocation for energy-efficient datacenters," in *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013, pp. 1345–1350.

[98] K. H. Kim, A. Beloglazov, and R. 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 (MGC 2009).* Champaign, USA: ACM, New York, USA, November 2009, pp. 1:1–1:6.

[99] K. H. Kim, R. Buyya, and J. Kim, "Power aware scheduling of Bag-of-Tasks applications with deadline constraints on dvs-enabled clusters," in *Proceedings of the 2007 IEEE Seventh International Symposium on Cluster Computing and the Grid (CCGrid 2007).*, May 2007, pp. 541–548.

[100] W. Kim, "Cloud computing architecture," *International Journal of Web and Grid Services*, vol. 9, no. 3, pp. 287–303, 2013.

[101] C. Klein, M. Maggio, K.-E. Årzén, and F. Hernández-Rodriguez, "Brownout: building more robust cloud applications," in *Proceedings of the 36th International Conference on Software Engineering*, July 2014, pp. 700–711.

[102] D. Kliazovich, P. Bouvry, and S. Khan, "GreenCloud: a packet-level simulator of energy-aware cloud computing data centers," *The Journal of Supercomputing*, vol. 62, no. 3, pp. 1263–1283, 2012.

[103] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. H. Katz, "NapSAC: Design and implementation of a power-proportional web cluster," in *Proceedings of the First ACM SIGCOMM Workshop on Green Networking (Green Networking 2010).* New York, USA: ACM, 2010, pp. 15–22.

[104] W. Lang and J. M. Patel, "Energy management for MapReduce clusters," *Proceedings of the Very Large Data Bases (VLDB) Endowment Journal*, vol. 3, pp. 129–139, 2010.

[105] J. Lawler, H. Howell-Barber, and A. Joseph, "A cloud computing methodology study of platform-as-a-service (paas) in the financial industry," *Journal OF Information Systems Applied Research*, 2015.

[106] Y. C. Lee, H. Han, A. Y. Zomaya, and M. Yousif, "Resource-efficient workflow scheduling in clouds," *Knowledge-based Systems*, vol. 80, pp. 153–162, 2015.

[107] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of hadoop clusters," *SIGOPS Operating Systems Review*, vol. 44, pp. 61–65, 2010.

[108] S.-H. Lim, B. Sharma, G. Nam, E. K. Kim, and C. Das, "Mdcsim: A multi-tier data center simulation, platform," in *Proceedings of the 2009 IEEE International Conference on Cluster Computing and Workshops*, August 2009, pp. 1–9.

[109] A. Mani Sankar, "Containers (Docker): A disruptive force in cloud computing." http://anandmanisankar.com/posts/container-docker-PaaS-microservices/, (Accessed on 02/25/2016).

[110] M. Mao and M. Humphrey, "A performance study on the vm startup time in the cloud," in *Proceedings of the IEEE Fifth International Conference on Cloud Computing (CLOUD 2012)*. Washington, DC, USA: IEEE Computer Society, 2012, pp. 423–430.

[111] D. Meisner, B. T. Gold, and T. F. Wenisch, "PowerNap: Eliminating server idle power," *SIGARCH Computer Architecture News*, vol. 37, no. 1, pp. 205–216, 2009.

[112] P. Mell and T. Grance, "The NIST definition of cloud computing," *Communications of the ACM*, vol. 53, no. 6, p. 50, 2010.

[113] X. Meng, C. Isci, J. Kephart, L. Zhang, E. Bouillet, and D. Pendarakis, "Efficient resource provisioning in compute clouds via VM multiplexing," in *Proceedings of the 7th International Conference on Autonomic Computing*, 2010, pp. 11–20.

[114] D. Merkel, "Docker: Lightweight Linux containers for consistent development and deployment," *Linux Journal*, vol. 2014, no. 239, 2014.

[115] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from Google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[116] V. Mohan Raj and R. Shriram, "Power aware provisioning in cloud computing environment," in *Proceedings of the 2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*, March 2011, pp. 6–11.

[117] R. Morabito, "Power consumption of virtualization technologies: an empirical investigation," in *Proceedings of the 2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)*, December 2015, pp. 522–527.

[118] I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, "An approach for characterizing workloads in Google cloud to derive realistic resource utilization models," in *Proceedings of the 7th IEEE International Symposium on Service Oriented System Engineering (SOSE 2013)*, March 2013, pp. 49–60.

[119] I. S. Moreno, R. Yang, J. Xu, and T. Wo, "Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement," in *Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on*, March 2013, pp. 1–8.

[120] N. Muthuvelu, C. Vecchiola, I. Chai, E. Chikkannan, and R. Buyya, "Task granularity policies for deploying bag-of-task applications on global grids," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 170–181, January 2013.

[121] G. Nagy, "Operating system containers vs. application containers," https://blog.risingstack.com/operating-system-containers-vs-application-containers/, May 2015, (Accessed on 10/22/2015).

[122] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 237–250.

[123] R. Nathuji and K. Schwan, "VirtualPower: Coordinated power management in virtualized enterprise systems," in *Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP 2007)*, October 2007, pp. 265–278.

[124] A. Nez, J. Vzquez-Poletti, A. Caminero, G. Casta, J. Carretero, and I. Llorente, "iCanCloud: A flexible and scalable cloud infrastructure simulator," *Journal of Grid Computing*, vol. 10, no. 1, pp. 185–209, 2012.

[125] S. Ostermann, K. Plankensteiner, R. Prodan, and T. Fahringer, "GroudSim: An event-based simulation framework for computational grids and clouds," in *Pro-*

ceedings of the 2010 European Conference on Parallel Processing (Euro-Par 2010) Workshops, August 2010, pp. 305–313.

[126] D. Pandit, S. Chattopadhyay, M. Chattopadhyay, and N. Chaki, "Resource allocation in cloud using simulated annealing," in *Proceedings of the 2014 Conference on Applications and Innovations in Mobile Computing (AIMoC 2014)*, February 2014, pp. 21–27.

[127] K. Park and V. S. Pai, "CoMon: A mostly-scalable monitoring system for planetlab," *SIGOPS Operating System Review*, vol. 40, no. 1, pp. 65–74, 2006.

[128] M. Pedram and I. Hwang, "Power and performance modeling in a virtualized server system," in *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW 2010)*, September 2010, pp. 520–526.

[129] D. Pelleg and A. W. Moore, "X-means: Extending K-means with efficient estimation of the number of clusters," in *Proceedings of 17th International Conference on Machine Learning (ICML 2000)*, 2000, pp. 727–734.

[130] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Proceedings of the 2009 Workshop on Energy-Efficient Design*, June 2009.

[131] D. T. Pham, S. S. Dimov, and C. Nguyen, "Selection of k in K-means clustering," *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, vol. 219, no. 1, pp. 103–119, 2005.

[132] J.-M. Pierson and H. Casanova, "On the utility of DVFS for power-aware job placement in clusters," in *Proceedings of the 17th International European Conference on Parallel Processing (Euro-Par 2011)*, 2011, pp. 255–266.

[133] I. Pietri and R. Sakellariou, "Energy-aware workflow scheduling using frequency scaling," in *Proceedings of the 43rd International Conference on Parallel Processing Workshops (ICCPW 2014)*, September 2014, pp. 104–113.

[134] R. Prez-Castillo, I. G.-R. de Guzmn, and M. Piattini, "Knowledge discovery Metamodel-ISO/IEC 19506: A standard to modernize legacy systems," *Computer Standards and Interfaces*, vol. 33, pp. 519 – 532, 2011.

[135] D. Price and A. Tucker, "Solaris Zones: Operating system support for consolidating commercial workloads," in *Proceedings of the 18th USENIX Conference on System Administration (LISA 2004)*, 2004, pp. 241–254.

[136] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC 2012)*, October 2012, pp. 7:1–7:13.

[137] C. Reiss, J. Wilkes, and J. L. Hellerstein, *Google cluster-usage traces: format+ schema*, Google Inc., November 2011.

[138] J. Rolia, A. Andrzejak, and M. Arlitt, "Automating enterprise application placement in resource utilities," in *Self-Managing Distributed Systems*, M. Brunner and A. Keller, Eds., 2003, vol. 2867, pp. 118–129.

[139] R. Rosen, "Resource management: Linux kernel namespaces and cgroups," *Haifux, May*, 2013.

[140] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[141] G. Schwarz, "Estimating the dimension of a model," *The Annals of Statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[142] B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in Google compute clusters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC 2011)*, October 2011, pp. 3:1–3:14.

[143] D. Shepherd, "Containers as a Service (CaaS) is the cloud operating system - i build the cloud," http://www.ibuildthecloud.com/blog/2014/08/19/containers-as-a-service-caas-is-the-cloud-operating-system/, (Accessed on 03/01/2016).

[144] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 5, pp. 45–57, 2002.

[145] G. Singh *et al.*, "Workflow task clustering for best effort systems with Pegasus," in *Proceedings of the 15th ACM Mardi Gras Conference (MG 2008)*, January 2008, pp. 9:1–9:8.

[146] I. Solis Moreno, P. Garraghan, P. Townend, and J. Xu, "Analysis, modeling and simulation of workload patterns in a large-scale utility cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 2, pp. 208–221, 2014.

[147] S. Spicuglia, L. Y. Chen, R. Birke, and W. Binder, "Optimizing capacity allocation for big data applications in cloud datacenters," in *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM 2015)*, 2015, pp. 511–517.

[148] I. Sriram, "SPECI, a simulation tool exploring cloud-scale data centres," in *Cloud Computing*, M. Jaatun, G. Zhao, and C. Rong, Eds. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 381–392.

[149] I. Stojmenovic, "Simulations in wireless sensor and ad hoc networks: matching and advancing models, metrics, and solutions," *Communications Magazine, IEEE*, vol. 46, no. 12, pp. 102–107, December 2008.

[150] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth, "Evaluation of delta compression techniques for efficient live migration of large virtual machines," *ACM Sigplan Notices*, vol. 46, no. 7, pp. 111–120, 2011.

[151] A. Tchana, N. Palma, I. Safieddine, D. Hagimont, B. Diot, and N. Vuillerme, "Software consolidation as an efficient energy and cost saving solution for a SaaS/PaaS cloud model," in *Proceedings of the 2015 European Conference on Parallel Processing (Euro-Par 2015)*, August 2015, pp. 305–316.

[152] M. Tighe, G. Keller, M. Bauer, and H. Lutfiyya, "DCSim: A data centre simulation tool for evaluating dynamic virtualized resource management," in *Proceedings of*

the 8th international conference on Network and service management (CNSM 2012) and workshop on systems virtualiztion management (SVM 2012), October 2012, pp. 385–392.

[153] L. Tomas, C. Klein, J. Tordsson, and F. Hernandez-Rodriguez, "The straw that broke the camel's back: Safe cloud overbooking with application brownout," in *Proceedings of the 2014 International Conference on Cloud and Autonomic Computing (ICCAC 2014)*, September 2014, pp. 151–160.

[154] L. Tomás and J. Tordsson, "An autonomic approach to risk-aware data center overbooking," *IEEE Transactions on Cloud Computing*, vol. 2, no. 3, pp. 292–305, 2014.

[155] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 3, pp. 260–274, 2002.

[156] R. Urgaonkar, U. Kozat, K. Igarashi, and M. Neely, "Dynamic resource allocation and power management in virtualized data centers," in *Proceedings of the 2010 IEEE Network Operations and Management Symposium (NOMS 2010)*, April 2010, pp. 479–486.

[157] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-Optimal scheduling in hybrid IaaS clouds for deadline constrained workloads," in *Proceedings of the 3rd International Conference on Cloud Computing*, July 2010, pp. 228–235.

[158] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah *et al.*, "Apache Hadoop YARN: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing (SOCC 2013)*. ACM, 2013, pp. 5:1–5:16.

[159] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proceedings of the 2009 Conference on USENIX Annual Technical Conference*, 2009, pp. 28–28.

[160] G. von Laszewski, L. Wang, A. Younge, and X. He, "Power-aware scheduling of virtual machines in DVFS-enabled clusters," in *Proceedings of the 2009 IEEE Inter-*

national Conference on Cluster Computing and Workshops (CLUSTER 2009)., August 2009, pp. 1–10.

[161] D. Wang, C. Ren, S. Govindan, A. Sivasubramaniam, B. Urgaonkar, A. Kansal, and K. Vaid, "ACE: abstracting, characterizing and exploiting peaks and valleys in datacenter power consumption," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, 2013, pp. 333–334.

[162] L. Wang, J. Tao, G. von Laszewski, and D. Chen, "Power aware scheduling for parallel tasks via task clustering," in *Proceedings of the IEEE 16th International Conference on Parallel and Distributed Systems (ICPADS 2010)*, December 2010, pp. 629–634.

[163] X. Wang, Y. Yao, X. Wang, K. Lu, and Q. Cao, "CARPO: Correlation-aware power optimization in data center networks," in *Proceedings of the 2012 IEEE International Conference on Computer Communications (INFOCOM 2012)*, March 2012, pp. 1125–1133.

[164] B. Wickremasinghe, R. Calheiros, and R. Buyya, "CloudAnalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications," in *Proceedings of the 24th IEEE International Conference on Advanced Information Networking and Applications (AINA 2010)*, April 2010, pp. 446–452.

[165] M. G. Xavier, M. V. Neves, and C. A. F. D. Rose, "A performance comparison of container-based virtualization systems for mapreduce clusters," in *Proceedings of the 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, February 2014, pp. 299–306.

[166] M. G. Xavier, F. D. Rossi, C. A. F. De Rose, R. N. Calheiros, and D. G. Gomes, "Modeling and simulation of global and sleep states in acpi-compliant energy-efficient cloud environments," *Concurrency and Computation: Practice and Experience*, 2016. [Online]. Available: http://dx.doi.org/10.1002/cpe.3839

[167] Yahoo, "Yahoo! expands its m45 cloud computing initiative," https://yodel.yahoo.com/blogs/product-news/yahoo-expands-m45-cloud-computing-initiative-5065.html, November 2010, (Accessed on 03/21/2016).

[168] E. Yaqub, R. Yahyapour, P. Wieder, A. Jehangiri, K. Lu, and C. Kotsokalis, "Metaheuristics-based planning and optimization for SLA-Aware resource management in PaaS clouds," in *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2014)*, December 2014, pp. 288–297.

[169] R. Yogamangalam and V. S. Sriram, "A review on security issues in cloud computing," *Journal of Artificial Intelligence*, vol. 6, no. 1, pp. 1–7, 2013.

[170] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Internet Services and Applications*, vol. 1, no. 1, pp. 7–18, 2010.

[171] Q. Zhang, J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in Google's compute clusters," in *Proceedings of the 5th International Workshop on Large Scale Distributed Systems and Middleware*, September 2011, pp. 1–6.

[172] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes, "CPI2: CPU performance isolation for shared compute clusters," in *Proceedings of the 8th ACM European Conference on Computer Systems (EuroSys '13)*, 2013, pp. 379–391.

[173] W. Zhao, Y. Peng, F. Xie, and Z. Dai, "Modeling and simulation of cloud computing: A review," in *Proceedings of the 2012 IEEE Asia Pacific Congress on Cloud Computing (APCloudCC 2012)*, November 2012, pp. 20–24.

[174] K. Zheng, X. Wang, L. Li, and X. Wang, "Joint power optimization of data center network and servers with correlation analysis," in *Proceedings of the 2014 IEEE International Conference on Computer Communications (INFOCOM 2014)*, April 2014, pp. 2598–2606.