Brownout-Oriented and Energy Efficient Management of Cloud Data Centers

Minxian Xu

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

November 2018

School of Computing and Information Systems
THE UNIVERSITY OF MELBOURNE

Brownout-Oriented and Energy Efficient Management of Cloud Data Centers

Minxian Xu
Principal Supervisor: Prof. Rajkumar Buyya

Abstract

Cloud computing paradigm supports dynamic provisioning of resources for delivering computing for applications as utility services as a pay-as-you-go basis. However, the energy consumption of cloud data centers has become a major concern as a typical data center can consume as much energy as 25,000 households. The dominant energy efficient approaches, like Dynamic Voltage Frequency Scaling and VM consolidation, cannot function well when the whole data center is overloaded. Therefore, a novel paradigm called brownout has been proposed, which can dynamically activate/deactivate the optional parts of the application system. Brownout has successfully shown it can avoid overloads due to changes in the workload and achieve better load balancing and energy saving effects.

In this thesis, we propose brownout-based approaches to address energy efficiency and cost-aware problem, and to facilitate resource management in cloud data centers. They are able to reduce data center energy consumption while ensuring Service Level Agreement defined by service providers. Specifically, the thesis advances the state-of-art by making the following key contributions:

1. An approach for scheduling cloud application components with brownout. The approach models the brownout enabled system by considering application components, which are either mandatory or optional. It also contains brownout-based algorithm to determine when to use brownout and how much utilization can be reduced.

2. A resource scheduling algorithm based on brownout and approximate Markov Decision Process approach. The approach considers the trade-offs between saved energy and the discount that is given to the user if components or microservices are deactivated.

3. A framework that enables brownout paradigm to manage the container-based environment, and provides fine-grained control on containers, which also contains several scheduling policies for managing containers to achieve power saving and QoS constraints.

4. The design and development of a software prototype based on Docker Swarm to reduce energy consumption while ensuring QoS in Clouds, and evaluations of different container scheduling policies under real testbeds to help service provider deploying services in a more energy-efficient manner while ensuring QoS constraint.

5. A perspective model for multi-level resource scheduling and a self-adaptive approach for interactive workloads and batch workloads to ensure their QoS by considering the renewable energy at Melbourne based on support vector machine. The proposed approach is evaluated under our developed prototype system.

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.

Minxian Xu, November 2018

Preface

This thesis research has been carried out in the Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne. The main contributions of the thesis are discussed in Chapters 2-7 and are based on the following publications:

- **Minxian Xu**, Amir Vahid Dastjerdi, and Rajkumar Buyya, "Energy Efficient Scheduling of Cloud Application Components with Brownout," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 1, Number 2, Pages: 40-53, ISSN: 2377-3782, IEEE Computer Society Press, USA, July-Dec 2016.

- **Minxian Xu**, Wenhong Tian, and Rajkumar Buyya, "A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing," *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 29, No. 12, Pages: 1-16, ISSN: 1532-0626, Wiley Press, New York, USA, June 25, 2017.

- **Minxian Xu** and Rajkumar Buyya, "Energy Efficient Scheduling of Application Components via Brownout and Approximate Markov Decision Process," *in Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC)*, LNCS, Springer-Verlag Press, Berlin, Germany), Málaga, Spain, November 13-16, 2017.

- **Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "iBrownout: An Integrated Approach for Managing Energy and Brownout in Container-based Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 4, Number 1, Pages: 53-66, ISSN: 2377-3782, IEEE Computer Society Press, USA, Jan-Mar 2019.

- **Minxian Xu** and Rajkumar Buyya, "Brownout Approach for Adaptive Management of Resources and Applications in Cloud Computing Systems: A Taxonomy and Future Directions," *ACM Computing Surveys (CSUR)*, Volume 52, No. 8, Pages: 1-27, ISSN: 0360-0300, ACM Press, New York, USA, February 2019.

- **Minxian Xu** and Rajkumar Buyya, "BrownoutCon: A Software System based on Brownout and Containers for Energy Efficient Clouds," *Journal of Systems and Software (JSS)*, 2019 (under review).
- **Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (under review).

Acknowledgements

Time flies in the blink of an eye. This year is the tenth year when I started my study at university. I hope this thesis is a new starting point of my academic study rather than an endpoint.

I am very thankful to have the opportunity to be supervised by Professor Rajkumar Buyya during my PhD candidature in the recent three years. I met him 5 years ago and then applied his PhD, it just looks like what happened yesterday. I would like to express my deepest gratitude for his continuous insights and supports for my PhD life.

I thank Dr. Amir Vahid Dastjerdi for his guidance during the first year of my PhD. I would also like to thank the members of my PhD committee: Prof. Liz Sonenberg and Dr. Adel Nadjaran Toosi. Prof. Liz has provided constructive comments and advice on my research. Dr. Adel has collaborated with me and helped me to improve my work.

Besides, I thank Dr. Marcos Assuncao, who kindly helped to provide the Grid'5000 infrastructure resources to support my experiments. I thank the visiting scholars to CLOUDS laboratory, Prof. Wenhong Tian and Prof. Satish Narayana Srirama. They have provided impressive views on enhancing my research. I express my thanks to the other post-docs in CLOUDS laboratory, Dr. Rodrigo Calheiros, Dr. Maria Rodriguez, Dr. Sukphal Singh Gill, Dr. Jungming Jay Son, who have also provided insights and suggestions on the technical side.

I would like to thank all members of the CLOUDS Laboratory. To my friends, Dr. Chenhao Qu, Dr. Bowen Zhou, Dr. Xunyun Liu, Safiollah Heidari, Caesar Wu, Sara Kardani Moghaddam, Muhammad H. Hilman, Redowan Mahmud, Muhammed Tawfiqul Islam, Shashikant Ilager for their friendship and support. I thank them to proof-reading my work and they gave me their useful comments.

I acknowledge the China Scholarship Council, University of Melbourne and ARC Discovery Project for providing me with scholarships to pursue my doctoral study.

My deepest gratitude goes to my family. I thank my parents and parents-in-law, without their encouragement, I cannot obtain current achievements. My special gratitude goes to my wife, Ms. Mengyun Liu, her sacrifice and love support me to reach here.

Minxian Xu
Melbourne, Australia
October 2018

Contents

List of Figures

List of Tables

Chapter 1

Introduction

CLOUD computing has gained significant attention from the academic and industry communities in recent years. It provides the vision that encompasses the movement of computing elements, storage, and software delivery away from the personal computer and local servers into the next generation computing infrastructure hosted by large companies such as Amazon Web Service (AWS), Microsoft Azure, and Google. Cloud computing has three distinct characteristics that differentiate it from its traditional counterparts: pay-as-you-go model, on-demand provisioning of resources, and elasticity [41].

Cloud computing offers three types of resources delivery models to users [107]: (i) Infrastructure as a Service (IaaS) which offers computing, network, and storage resources, (ii) Platform as a Service (PaaS) which provides users tools that facilitate the deployment of cloud applications, and (iii) Software as a Service (SaaS) which enables users to run the provider's software on the cloud infrastructure.

The aim of cloud computing is providing resources in the form of utility like water, gas, and electricity for daily use. Some attractive characteristics including on-demand resource provisioning model, scalability enhancement, operational cost reduction, and convenient access are offered by Clouds. All these features enable cloud computing to be appealing to business runners, which gets rid of the complexity for service providers to do provisioning plan and allows companies to begin with the minimum resources as required. Cloud platforms like EC2, Google Cloud, and Azure, have been built by large infrastructure providers to support applications around the world with the purpose of assuring these applications to be scalable and available for the demands of the users [107]. The cloud customers are allowed to dynamically lease and unlease the on-demand resources based on their requirements.

Although cloud data centers are providing compelling features for customers, the energy consumption of data centers has become a major topic of research. Fig. 1.1[1] illustrates the past and predicted growth rate of total power usage of U.S. data center from 2000 to 2020. It also shows the energy growth of data centers can be quite fast if the industry does not make any further efficiency improvements after 2010. The servers hosted in data centers dissipate heat and need to be maintained by cooling infrastructure, which provides the cooling resource to extract the heat from IT devices. Though the cooling infrastructure is already efficient to some extent, the servers are still one of the major energy consumers.

Due to the rapid growth of the Internet of Things (IoT) based applications, the amount of cloud services is increasing exponentially, which further increases the power consumptions of cloud data centers by 20-25% every year [90]. The data centers in U.S. consumed 100 billion kilowatt hours (kWh) in 2015, which is sufficient for Washington City [132] [29][50]. The consumption of electricity will reach 150 billion kWh by 2022 i.e. increase by 50% [126]. Energy consumption in cloud data centers can grow to 8000 terawatt hours (TWh) in 2030 if controlled mechanisms are not identified [22][26]. Because of the underutilized and overload of resources in infrastructure (cooling, computing, storage, networking etc.), the energy consumption in cloud data centers is not efficient and mostly the energy is consumed while some of the resources are in idle state, which increases the cost of cloud services [132].

Cloud data centers not only consume huge energy consumption but also have a non-negligible impact on the environment [6]. It is reported that 78.7 million metric tons of CO2 are emitted by data centers, which equals the 2% of global emissions [110][150]. It is also estimated that cloud data centers will use 20% of the world's electricity and emit up to 5.5% of global carbon footprint by 2025, which consume more energy than most countries [93]. Recently, some dominant service providers established a community to promote energy efficiency for data centers to minimize the impact on the environment, which is also known as Green Grid [33]. Carbon footprints produced by cloud data centers is same as aviation industry [38]. In the current scenario, the service providers are

[1]https://www.datacenterknowledge.com/archives/2016/06/27/heres-how-much-energy-all-us-data-centers-consume

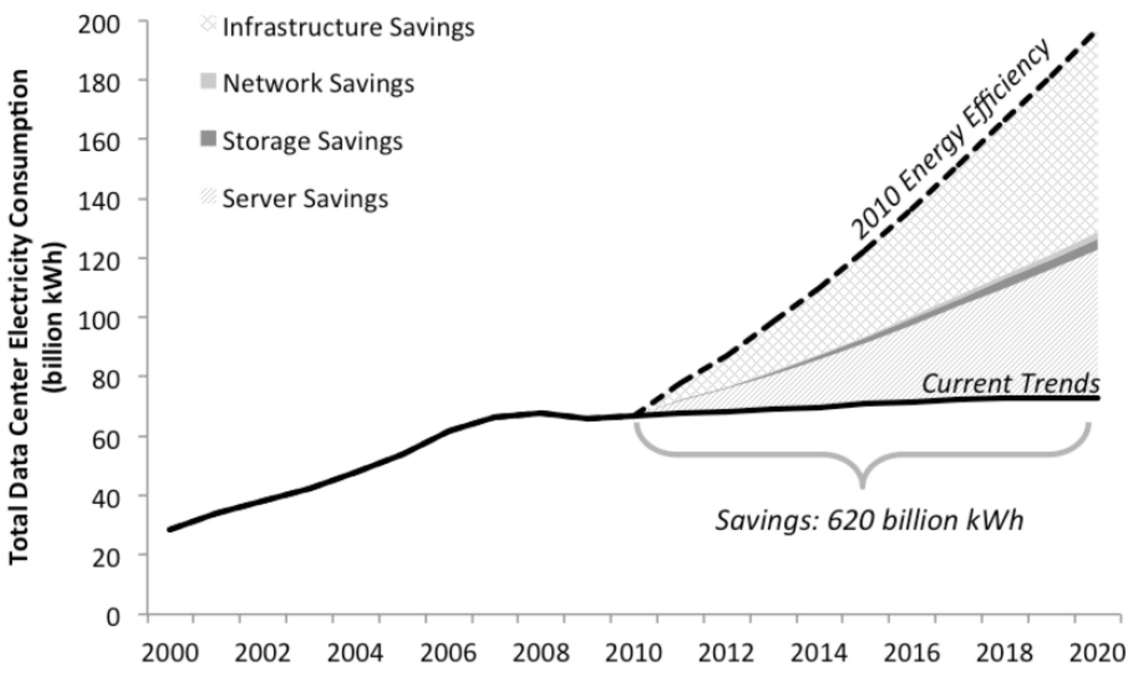


Figure 1.1: Power consumption of data centers

looking for alternative ways to reduce the carbon footprint of their infrastructure. The prominent cloud providers such as Google, Amazon, Microsoft and IBM have assured to attain zero production of carbon footprints and aimed to find the new ways to make data centers and cloud-based services eco-friendly [133]. Thus, cloud data centers needs to provision cloud services that minimize carbon footprint and reduce heat release in the form of greenhouse gas emissions [133].

To solve the above challenges of energy-efficient cloud data centers, a large body of researchers have proposed resource scheduling models, algorithms and architectures. However, energy efficiency is still a challenge for future researchers. To ensure high-level of sustainability, holistic management of resources can solve new open challenges existing in resource scheduling. Methods are required to harness renewable energy to reduce carbon footprints without the use of coal-based resources. Additionally, cooling expenses can be minimized by developing waste heat utilization and free cooling mechanisms. A promising way is based on location-aware ideal climatic conditions, which can achieve the efficient implementation of free cooling and renewable energy production techniques [69]. Further, waste heat recovery locations are required to be identified for an efficient implantation of waste heat recovery predictions. Cloud providers such as Google, Amazon, IBM, Facebook, and Microsoft are utilizing more green energy resources instead of grid electricity [156].

1.1 Motivations

Energy efficiency in cloud data centers is a challenging objective due to applications and data are growing fast and complex [96]. Normally, the applications and data are required to be processed within the required time, thus, large and powerful servers are required to offer services. To ensure the sustainability of future growth of data centers, cloud data centers must be designed to be efficiently utilize the resources of infrastructure and minimize energy consumption. To address this problem, the concept of Green Cloud is proposed, which aims to manage cloud data centers in an energy efficient manner [33]. Consequently, data centers are required to offer resources while satisfying Quality of Service (QoS), as well as reduce energy consumption.

One of the main reasons of high energy consumption of cloud data centers lies in that computing resources are inefficiently utilized by applications on servers [145][124]. RightScale states that the cloud consumers waste between 30-45% of their total cloud consumption [46]. Therefore, applications are currently built with microservices to utilize infrastructure resource more efficiently. Microservices are also referred to a set of self-contained application components [116]. The components encapsulate its logic and expose its functionality via interfaces to make them flexible to be deployed and replaced. With microservices or components, developers and user benefit from their technological heterogeneity, resilience, scalability, ease of deployment, organizational alignment, composability and optimization for replicability. It also brings the advantage of more fine-grained control over the application resource usage.

Thus, in this thesis, we take advantage of **brownout**, a paradigm inspired from voltage shutdown that copes with emergency cases. In the original brownout scenario, the light bulbs emit fewer lights to save energy consumption. In Cloud scenario, brownout can be applied to microservices or application components that are allowed to be temporarily deactivated without affecting main functionality. When brownout is triggered, the user's experience is temporally degraded to relieve the overloaded situation and reduce energy consumption.

It is common for microservices or application components to have this brownout feature. Klein et al. [85] introduced an online shopping system that has a recommendation engine to recommend products to users. The recommendation engine enhances the function of the whole system, while it is not necessary to keep it running all the time, especially under the overloaded situation. As the recommendation engine requires more resource in comparison to other components, if it is deactivated, more clients with essential requests or QoS constraints can be served. Apart from this example, brownout paradigm is also suitable for other systems that allow some microservices or application components to not keep running all the time.

To study service providers' requirement and concerns for managing services based on containers, we give a motivation example of a real-world case study with brownout technology. A good example of the microservice-based system is the web-based service. An online shopping system implemented with microservices is presented in [2], which contains multiple microservices, including user, user database, payment, shipping, front-end, orders, carts, catalog, carts database and etc. As it is implemented with microservices, each microservice can be activated or deactivated independently. When requests are bursting, the online shopping system may be overloaded, and it cannot satisfy QoS requirements. To handle the overloads and reduce energy consumption, the brownout approach can be applied to temporarily disable some microservices, such as the recommendation engine, to save resource and power. By deactivating the recommendation engine, the system is able to serve more requests with the essential requirement and satisfy QoS. When the system is not overloaded anymore, the disabled microservices are activated again. Considering the overloaded situation, we assume that the service provider of this online shopping system is interested to improve QoS and save energy costs. In addition, the service provider may prefer to apply brownout to manage microservices in their systems.

1.2 Research Problems and Objectives

This thesis focuses on self-adaptive resource scheduling for applications in cloud computing systems based on brownout approaches. The optimization objectives are energy and required QoS. In respect to this objective, the following research problem is investigated:

Designing algorithms for energy-efficient and cost-effective scheduling of multiple applications in Clouds with distributed and heterogeneous resources under quality of service constraints.

The research problem can consist of several sub-problems:

- **How to deal and manage resource allocation for user requests under overloaded situation?** Since unpredicted workloads, like request bursts, can trigger system performance degradation and lead to overloads, it is necessary to handle the overloads and allocate resources to satisfy the QoS for users.

- **How to monitor application status and make self-adaptive decisions under different workloads?** Predicting workloads accurately is helpful for designing resource scheduling policies for energy-efficient and cost-effective purposes. It is important to have a mechanism to predict workloads tendency.

- **How to reduce energy consumption while ensuring QoS?** Reducing energy consumption is one of the main objectives. However, generally, there are trade-offs between saved energy and QoS. The scheduling policies should balance the trade-offs between energy and QoS, like the response time of services.

- **How to identify the optional components/microservices?** In brownout enabled model, the components or microservices are identified as optional and mandatory ones, and only the optional ones can be deactivated temporarily to reduce resource utilization and energy consumption. It is required to identify which components/microservices are the optional ones.

- **When to trigger the brownout mechanism?** Brownout should only be triggered under certain situations, for example, the system is overloaded, since brownout comes along with performance degradation or functions deactivation. An effective mechanism to trigger brownout is also needed.

- **How to select the components/microservices to be deactivated temporarily?** The ideal design is to find an optimal list of optional components and deactivate them, which minimizes the energy consumption while ensuring the QoS. Thus, a component/microservice selection algorithm is required to achieve this goal. If the algorithm is not optimal, alternatively, a satidfactory algorithm should be competitive with the optimal algorithm in terms of time and cost.

To tackle the above research problems and challenges, the following objectives have been identified:

- Conduct a comprehensive survey and review of brownout-based approaches for cloud computing systems to identify the existing gaps in this area.
- Define a system model that enables brownout by considering application components, which are either mandatory or optional.
- Propose brownout enabled algorithm to determine when to use brownout and how much utilization can be reduced.
- Develop approaches that consider the trade-offs between discount that should be given to user if a component is deactivated and how much energy can be saved.
- Propose an approach based on microservices to reduce energy consumption while ensuring QoS in Clouds.
- Design and develop a software prototype based on microservices management to reduce energy consumption while ensuring QoS in Clouds.

1.3 Methodology

This thesis endeavors to reduce the energy consumption of data centers while satisfying the QoS defined by users. To achieve this goal, the following methodology for energy management problem has been adopted as shown in Figure 1.2

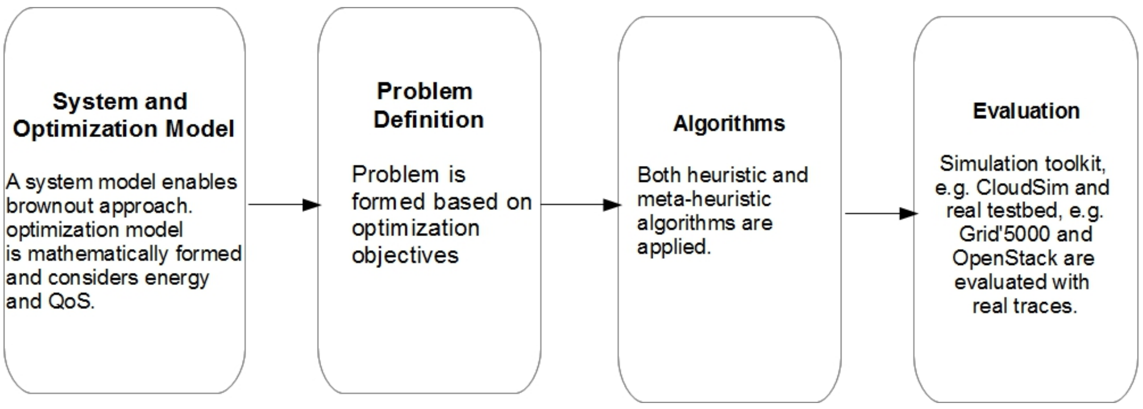

System and Optimization Model
A system model enables brownout approach. optimization model is mathematically formed and considers energy and QoS.
Problem Definition
Problem is formed based on optimization objectives
Algorithms
Both heuristic and meta-heuristic algorithms are applied.
Evaluation
Simulation toolkit, e.g. CloudSim and real testbed, e.g. Grid'5000 and OpenStack are evaluated with real traces.

Figure 1.2: The methodology used in the thesis

1. **System and optimization model.** We define a system model that enables the brownout approach, in which the application components are defined as mandatory or optional. Based on the system model, we mathematically define the optimization model that considers energy consumption and QoS defined by users.

2. **Problem definition.** We formally define the optimization problem based on the optimization objectives in the form of mathematical formula with the energy and predefined QoS.

3. **Algorithms.** The algorithms used to solve the optimization problems are based on heuristic and meta-heuristic algorithms. The time complexity of the proposed algorithms is also evaluated and analyzed.

4. **Evaluation.** The approaches in this thesis were evaluated both in simulators (CloudSim [42] developed by CLOUDS lab from University of Melbourne) and real testbeds (Grid'5000 from INRIA [67], OpenStack [15] at CLOUDS lab). The workloads used in this thesis were derived from real traces, including Wikipedia [151] and PlanetLab [122]. The major parameters we have evaluated are energy consumption and QoS (discount, response and SLA violations). The scalability of prototype system in Chapter 5 is also discussed.

1.4 Contributions

The contributions of this thesis can be broadly categorized into (i) A taxonomy and review of the brownout-based approaches in self-adaptive area, (ii) Scheduling algorithms for application components with brownout to balance the energy consumption and discount given to user, (iii) A meta-heuristic algorithm based Markov Decision Process to select application components to improve the trade-offs between energy consumption and discount, (iv) An approach based on brownout and containers to save power consumption while ensuring QoS, (v) A prototype system based on Docker Swarm to support brownout evaluated performance. The detailed **key contributions** of the thesis are as follows:

1. A review and taxonomy of brownout-based adaptive management of resources and applications for cloud computing systems.

2. Scheduling cloud application components with brownout :

    - A system model considers application components, which are either mandatory or optional, and presenting brownout enabled algorithm to determine when to use brownout and how much utilization can be reduced.
    - An approach considers the trade-offs between discount that should be given to user if a component is deactivated and how much energy can be saved.
    - A number of policies that consider the aforementioned trade-offs and dynamically make decisions on which components are going to be deactivated.

3. Scheduling of application components via brownout and approximate Markov Decision Process :

    - An approach based on brownout-based approximate Markov Decision Process to improve the trade-offs between energy and discount.
    - A model considers the trade-offs between saved energy and the discount that is given to user if components or microservices are deactivated.
    - An efficient algorithm based on brownout and approximate Markov Decision Process to achieve the better trade-offs than baselines.

4. Managing energy and brownout in container-based clouds :

    - An approach based on containers to reduce energy consumption while ensuring QoS in Clouds.
    - An effective architecture that enables brownout paradigm to mange the container-based environment, which enables fine-grained control on containers. .
    - Several scheduling policies for managing containers to achieve power saving and QoS constraints.
    - Evaluations in INRIA Grid′5000 testbed for Wikipedia workload.

5. A software system based on brownout and containers for clouds :

    - A software prototype based on Docker Swarm to reduce energy consumption while ensuring QoS in Clouds.
    - An effective system model that enables brownout approach to manage the containers and resources in a fine-grained manner.
    - Designing and implementing a software system based on Docker Swarm to provide energy-efficient approaches for cloud data centers.
    - Evaluation of different container scheduling polices under real testbeds to help service provider deploying services in a more energy-efficient manner while ensuring QoS constraint.

6. A self-adaptive approach for managing applications and harnessing renewable energy: :

    - Providing a perspective model for multi-level adaptive resource scheduling to manage workloads and renewable energy;
    - Proposing a self-adaptive approach for interactive workloads and batch workloads to ensure their QoS by considering the predicted renewable energy at Melbourne based on support vector machine;
    - Implementing a prototype system derived from the perspective model and the proposed approach on a small-scale testbed with 8 hosts;

- Evaluating the performance of the self-adaptive approach in the proposed prototype system.

1.5 Thesis Organization

The core chapters of this thesis are mostly derived from the publications made during my PhD candidature. Fig. 1.3 shows the structure of the thesis, and details are described as below:

- Chapter 2 presents a review and taxonomy of brownout-based adaptive resources and applications management for cloud computing systems. This chapter is partially derived from:
  - **Minxian Xu** and Rajkumar Buyya, "Brownout Approach for Adaptive Management of Resources and Applications in Cloud Computing Systems: A Taxonomy and Future Directions," *ACM Computing Surveys (CSUR)*, Volume 52, No. 8, Pages: 1-27, ISSN: 0360-0300, ACM Press, New York, USA, February 2019.
  - **Minxian Xu**, Wenhong Tian, and Rajkumar Buyya, "A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing," *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 29, No. 12, Pages: 1-16, ISSN: 1532-0626, Wiley Press, New York, USA, June 25, 2017.
- Chapter 3 proposes a scheduling algorithm for application components with brownout to balance the energy consumption and discount given to user. This chapter is derived from:
  - **Minxian Xu**, Amir Vahid Dastjerdi, and Rajkumar Buyya, "Energy Efficient Scheduling of Cloud Application Components with Brownout," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 1, Number 2, Pages: 40-53, ISSN: 2377-3782, IEEE Computer Society Press, USA, July-Dec 2016.

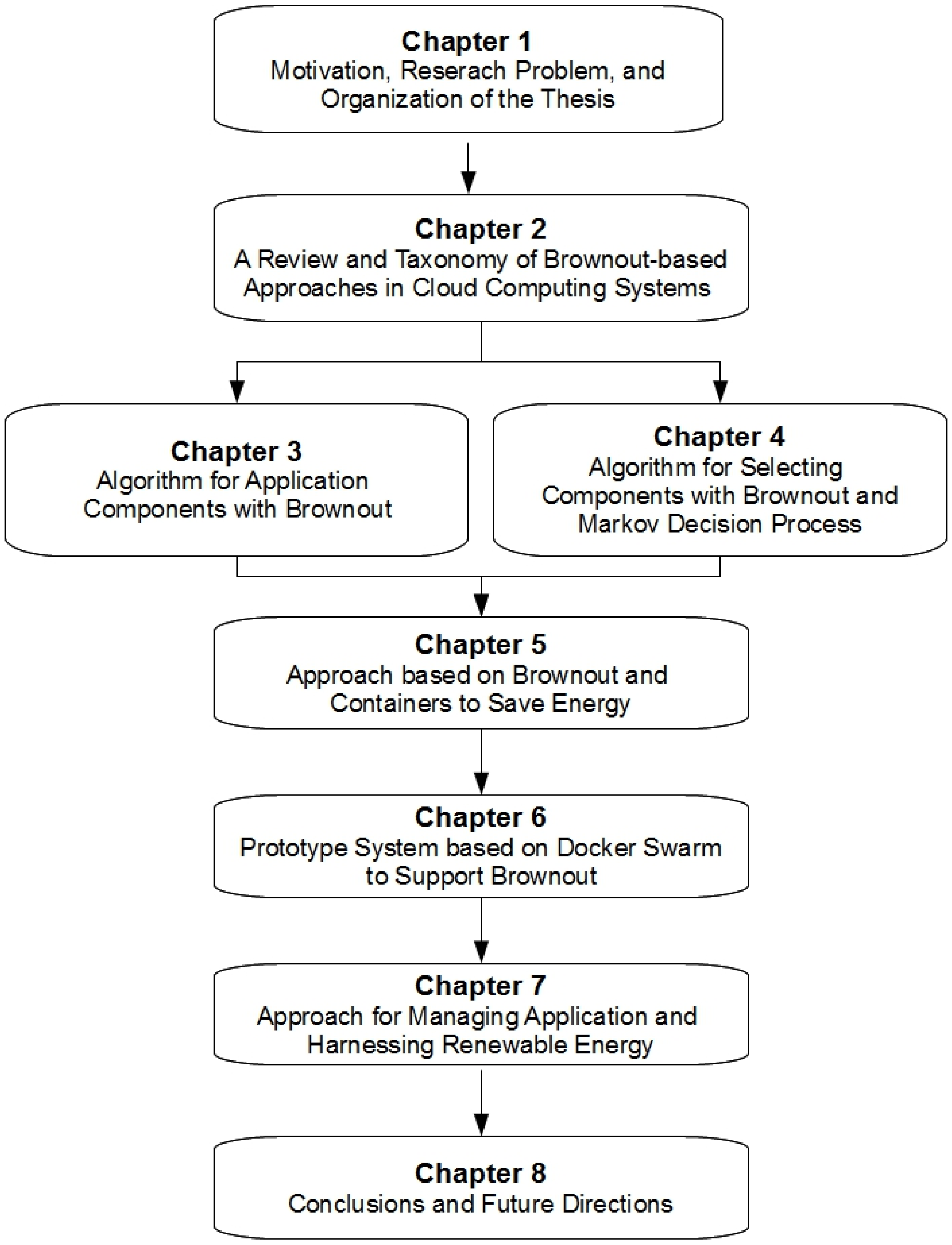

Chapter 1
Motivation, Reserach Problem, and Organization of the Thesis
Chapter 2
A Review and Taxonomy of Brownout-based Approaches in Cloud Computing Systems
Chapter 3
Algorithm for Application Components with Brownout
Chapter 4
Algorithm for Selecting Components with Brownout and Markov Decision Process
Chapter 5
Approach based on Brownout and Containers to Save Energy
Chapter 6
Prototype System based on Docker Swarm to Support Brownout
Chapter 7
Approach for Managing Application and Harnessing Renewable Energy
Chapter 8
Conclusions and Future Directions

Figure 1.3: The thesis organization

- Chapter 4 presents a meta-heuristic algorithm based Markov Decision Process to select application components to improve the trade-offs between energy consumption and discount. This chapter is derived from:
  - **Minxian Xu** and Rajkumar Buyya, "Energy Efficient Scheduling of Application Components via Brownout and Approximate Markov Decision Process," *Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC)*, LNCS, Springer-Verlag Press, Berlin, Germany), Malaga, Spain, November 13-16, 2017.
- Chapter 5 investigates approach based on brownout and containers to save power consumption while ensuring QoS. This chapter is derived from:
  - **Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "iBrownout: An Integrated Approach for Managing Energy and Brownout in Container-based Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 4, Number 1, Pages: 53-66, ISSN: 2377-3782, IEEE Computer Society Press, USA, Jan-Mar 2019.
- Chapter 6 introduces a prototype system based on Docker Swarm to support brownout and evaluate performance. It is derived from:
  - **Minxian Xu** and Rajkumar Buyya, "BrownoutCon: A Software System based on Brownout and Containers for Energy Efficient Clouds," *Journal of Systems and Software (JSS)*, 2019 (under review).
- Chapter 7 presents a self-adaptive approach for managing applications and harnessing renewable energy for cloud data centers. It is derived from:
  - **Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (under review).

- Chapter 8 concludes the thesis with a summary of the key research outcomes and a discussion of future directions.

Chapter 2

Taxonomy and Literature Review

Cloud computing is viewed as an emerging approach for provisioning resources and managing applications. It provides attractive features, such as on-demand model, scalability enhancement, and management costs reduction. However, cloud computing systems continue to face problems such as hardware failures, overloads caused by unexpected workloads, or the waste of energy due to inefficient resource utilization, which all result to resource shortages and application issues such as delays or saturated eventually. A paradigm named brownout has been applied to handle these issues by adaptively activating or deactivating optional parts of applications or services to manage resource usage in the cloud computing system. Brownout has successfully shown it can avoid overloads due to changes in the workload and achieve better load balancing and energy saving effects. In this chapter, we propose a taxonomy of brownout-based approach for managing resources and applications adaptively in cloud computing systems and carries out a comprehensive survey.

2.1 Introduction

CLOUD computing has been regarded as one of the most dominant technologies that promote the future economy [79]. Traditionally, the service providers used to establish their own data centers with a huge investment to maintain the applications and provide services to users. With cloud computing, resources can be leased by application providers and the applications can be deployed without any upfront costs. Nowadays, many applications are developed for the cloud computing systems, and Clouds also provide elastic resources for applications [136]. This feature attracts enterprises to migrate

This chapter is derived from:
• **Minxian Xu** and Rajkumar Buyya, "Brownout Approach for Adaptive Management of Resources and Applications in Cloud Computing Systems: A Taxonomy and Future Directions," *ACM Computing Surveys (CSUR)*, Volume 52, No. 8, Pages: 1-27, ISSN: 0360-0300, ACM Press, New York, USA, February 2019.
• **Minxian Xu**, Wenhong Tian, and Rajkumar Buyya, "A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing," *Concurrency and Computation: Practice and Experience (CCPE)*, Volume 29, No. 12, Pages: 1-16, ISSN: 1532-0626, Wiley Press, New York, USA, June 25, 2017.

their local applications to Clouds [25].

In addition to the traditional requirements, the cloud applications are experiencing unpredictable workloads because of the dynamic amount of requests and users [44]. Thus, cloud computing systems are also required to be designed as robust to handle unexpected events, such as request bursts, which is commonly named as flash crowds that can increase the size of requests significantly. For example, the servers of Weibo, a Chinese web social network website owned by Sina, got a breakdown after a Chinese celebrity announced his new relationship, which resulted from the celebrity's fans flooding into the website [12].

Similarly, in cloud data centers, unexpected hardware failures are also common issues. In 2016, the severe weather led to the outage of Amazon data centers in Sydney, knocking the services of many companies to be offline [7]. Moreover, performance inference resulted from co-located applications and workload consolidations may lead to unexpected performance degradations [153].

2.1.1 Need for Adaptive Management in Cloud Computing Systems

To handle the aforementioned phenomena, applications are needed to be carefully designed and deployed. For instance, techniques such as auto-scaling [99], data replication [109], workload consolidation [76] and dynamic load balancing [121] are applied to overcome unexpected events only if the available resources are adequate. However, these unexpected events are generally only lasting a relatively short duration, it is not economical to provision sufficient capacity as much as possible. Without sufficient resource provisioning, the applications can be saturated, and cause the users to experience longer response time or even no response at all. As a result, service providers may lose customers and revenues. Therefore, we argue that the adaptive management of resources and applications is needed for cloud computing systems.

With the adaptive management of resources and applications, different benefits can be achieved. Adaptive management can improve the Quality of Service (QoS) guarantee of cloud services. QoS guarantee plays a crucial role in system performance for Clouds environment [89], and cloud computing systems are required to be designed to offer QoS guaranteed services [134] [135]. It is a challenging issue for Clouds to support various co-located applications with different QoS constraints, and provision resources efficiently to be adaptive to the users' dynamic behaviors. These aspects make it difficult to provision resources efficiently [130] [56]. It is reported that 53% mobile users abandon pages that have no response within three seconds. Therefore, Google firmly recommended a one-second web page loading time to improve user's experience [60]. QoS of applications can be improved by dynamically adding or removing hosts via adaptive management.

Energy efficiency of cloud computing systems can be improved by adaptive management. It has become a major problem in the IT industry that huge energy is consumed by cloud data centers [104]. The rise and evolution of complicated computation intensive applications have promoted the establishment of large cloud data centers that boost the total amount of power usage [32]. The physical servers deployed in Clouds generate massive heat and require to be managed in an environment equipped with powerful air conditioners. One of the key reasons for huge energy usage is due to the inefficient utilization of resources [33]. Adaptive management is able to improve resource usage so that energy consumption can be reduced. For example, when there are fewer requests, adaptive management can reduce energy consumption by consolidating workloads onto fewer active physical machines, thus the idle physical machines can be turned into the low-power mode or fully turned-off.

Balancing the loads in cloud computing systems is another objective that can be fulfilled by adaptive management. Load balancers are the regular components of web applications, which allow the system to be scalable and resilience [128]. Numerous load balancing algorithms have been introduced by researchers, focusing on different optimization targets, ranging from balancing virtual machines load to physical machines , with specific optimizations by both heuristic and meta-heuristic algorithms.The purposes of load balancing can be various, including geographical balancing [98], electricity costs reduction [127] and applications load balancing in cloud computing systems [108]. Adaptive management can avoid overloads to achieve load balancing.

A promising approach for adaptive management of resources and applications in cloud computing systems is brownout [85]. In the field of brownout, the applications/services are extended to have two parts: mandatory and optional. The mandatory parts are desired to keep running all the time, such as the critical services in the systems including data-relevant services. The optional parts, on the other hand, are not necessary to be active all the time and can be deactivated temporarily to ensure the system performance in the case of flash crowds.

A motivational example of brownout-enabled application is the E-Commerce system, where product descriptions are shown along with related products suggested to the end users. These related products are managed by a recommendation engine in the system. The recommendation engine can be identified as an optional part, because it is not strictly necessary for the core function to work. Indeed, when the system is overloaded, even if the recommendation engine improves the user experience, it is preferable to deactivate the engine temporarily to obtain a more responsive website for more users.

2.1.2 Motivation of Research

Currently, brownout approaches have been applied in cloud computing system for different optimization objectives, including system robustness improvement, overbooking, load balancing and energy efficiency. Therefore, we like to investigate them in depth as noted below:

- Brownout approach has shown a promising direction to manage applications and resources in cloud computing systems. Therefore, this article discusses the development and application of brownout approaches in the cloud computing area.
- We identify the necessity of a literature review to summarize the progress in brownout approach with adaptive management of resources and applications for cloud computing systems. Consequently, we have surveyed the existing articles relevant to this topic and aimed to draw more attention and efforts to advance research with brownout approaches.

2.1.3 Our Contributions

The major contributions of this chapter are summarized as follows:

- We propose a taxonomy of brownout-based adaptive management of resources and applications in cloud computing systems.

- We investigate a comprehensive review of brownout approaches in cloud computing systems for adaptive management of applications and resources.
- We categorize the studied brownout approaches according to their common features and properties, and compare the advantages and limitations of each approach.
- We identify the research gaps and future research directions in brownout-enabled cloud computing systems.

2.1.4 Related Surveys

A few articles have conducted surveys or taxonomies on resource management in cloud computing. Kaur et al.[81] conducted a comprehensive taxonomy and survey for energy efficient scheduling approaches in Clouds. Kong et al. [87] discussed the energy saving techniques from green energy perspective. Weerasiri et al. [149] introduced a survey and taxonomy for resource orchestration in Clouds while not focusing on adaptive resource management. Zhan et al. [152] investigated the cloud computing resource scheduling approaches and summarized their evolution. Mansouri et al. [102] presented a survey and taxonomy on resource management in Cloud environment, with the focus on management of storage resources. Singh et al. [135] proposed a systematic review of Quality of Service aware automatic resource scheduling approaches in Clouds scenario.

However, there is no existing survey and taxonomy focusing on brownout approach. As a complimentary, our article enhances previous surveys and focuses on the brownout-based approach. It also identifies the open challenges and future research directions in the area that applying brownout in cloud computing systems for adaptive management of resources and applications.

2.2 Background

In this section, we briefly introduce the background on cloud computing and adaptive management.

2.2.1 Cloud Computing

The appearance of cloud computing is regarded as a novel paradigm in information technology industry [39]. The objective of cloud computing is provisioning resources as utilities like water, gas, and electricity for daily use, which can be used on-demand. In addition, cloud computing also has the attractive features including scalability improvement, lower operational cost, and convenient access. All these features enable cloud computing to be appealing to business users, which gets rid of the complexity for service providers to design provisioning plan and allows companies to begin with the minimum resources as required. Large cloud service providers like Amazon, Google, and Microsoft, have established infrastructures to support services distributed in the world with the purpose of assuring these services to be scalable and available for the demands of the users [107]. The cloud customers are also benefiting from cloud computing that on-demand resources are allowed to be dynamically leased and unleased based on their requirements.

2.2.2 Adaptive Management

In the past years, a considerable amount of research in adaptive techniques to manage resources in the system has been conducted. The properties of adaptive techniques are achieving management in a self-adaptation manner, including protection, optimization, and recovery. These properties are often featured as self-* characteristics [113].

The feedback loop is the essential concept used to develop an adaptive system, which monitors the status and the environment of the system, and adapts as desired to obtain the required self-* characteristics. It is viewed as an important factor to enable adaptive management of resources and applications by taking advantage of feedback loops for applications [78]. In adaptive systems, one favorite representation of the feedback loop is the Monitor, Analyze, Plan, Execute and Knowledge loop, which is abbreviated as MAPE-K [23]. In the MAPE-K loop, there are several phases to be accomplished in the loop as below:

(1) Monitoring the system status and the environment situation by Monitor phase;

(2) Analyzing the collected data and determining whether the adaptation is required or not by Analyze phase;

(3) Planning the approach to adapt the system by Plan phase;

(4) Executing the plan by Execute phase,

where the knowledge pool is shared by these 4 phases and acts as integration role [49].

2.3 Article Selection Methodology

In this section, we introduce the approach we followed to find our surveyed articles as well as the outcome.

2.3.1 Source of Articles

Related articles are broadly searched in main-stream academic databases, including IEEEXplore, Springer, Elsevier, ACM Digital Library, ScienceDirect, Wiley Interscience and Google Scholar.

2.3.2 Search Method

Two phases of our search are involved. In the first phase, we use the keywords ”Brownout” and ”Cloud Computing” to search the title and abstract of research articles. Several results are found, however, the number of these articles are quite limited. We think that some articles may be motivated by the mechanism of brownout while they do not use the keywords in the title or abstracts. Thus, in the second phase, based on the initial brownout researches conducted in 2014, we plan to find other articles using brownout inspired by these work. We show the pioneering work of brownout approach in 2014 and their number of citations[2] in Table 2.1. We investigated the articles that cite these four papers and found more articles that are using brownout.

2.3.3 Outcome

We have found 18 research articles focusing on brownout approach in cloud computing systems for adaptive management of resources and applications. 77.8% of these research

[2]This search was conducted on Feb 5, 2018.

Table 2.1: The earliest work in brownout approach for cloud computing systems in 2014 and their citations

Title	Citations
”Brownout: Building more robust cloud applications”	84
”The straw that broke the camel’s back: safe cloud overbooking with application brownout”	22
”Improving cloud service resilience using brownout-aware load-balancing”	23
”Control-theoretical load-balancing for cloud applications with brownout”	14

papers were presented in conferences, 16.6% were published in journals and 5.6% in symposiums. Moreover, one master thesis [53] and one PhD thesis [113] have been explored for this topic. The contents of these two theses are derived from the research articles we have found and reviewed, therefore, they are not included in the following taxonomy and review.

2.4 Brownout Approach

Brownout is inspired by the blackout in emergency cases that to cope with electricity. Such as light bulbs emit fewer lights to save energy usage to handle emergency situations [142]. In this section, we introduce the background and evolution of brownout approach.

2.4.1 Overview of Brownout Approach

2.4.1.1 Definition

Brownout is a self-adaptive paradigm that enables or disables optional parts (application components or services) in the system concerning how to handle unpredictable workloads [85]. The idea behind the brownout paradigm is as follows: to be adaptive, optional parts might temporarily be deactivated so that the essential functions of the system is ensured as well as avoiding saturated applications. Deactivating certain optional functions can contribute to increasing request acceptance rate by utilizing the optional resources for the core functions.

Therefore, the brownout paradigm is a method to make cloud computing system to be adaptive the changing workloads. Brownout also enables to improve resource utilization while keeping applications responsive to avoid overloads. Moreover, brownout can be regarded as a special type of per request admission control, where some requests are fully admitted, while others may only be admissible without optional services. In the brownout paradigm, **dimmer** is a control knob that takes a value in [0, 1] to represent the probability to run the optional parts and manages the brownout activities in the system. [85].

2.4.1.2 Architecture Model

The brownout-based scheduling of resources and applications in cloud computing systems complies the conventional type of adaptive architectures. It is derived from MAPE-K [23] feedback loop control including phases, like the monitor, analyze, plan, execute, and knowledge illustrated in Figure 2.1. Cloud computing system is the target system of MAPE-K loop and combined with MAPE-K loop through sensors and actuators. The responsibilities of each module in MAPE-K are as below:

- **Knowledge module:** It is a module that describes the whole system at the abstract level by capturing the major system features and status. The captured information is applied to trigger the desirable adaptations;
- **Monitor module:** It is used to collect the data from sensors and monitor the system status continuously. The collected data is transferred to Analyze module for further analysis;
- **Analyze module:** It analyzes the data obtained from Monitor module and provides references for Plan module. Different analysis methods can be applied in this module;
- **Planning module:** It makes plans to change system states to be adaptive the fluctuations of workloads;
- **Execution module:** It implements the plans. Its main role is ensuring the specified system requirement. In addition, through actuators, it also traces the new changes and makes other plans based on the predefined rules in the knowledge pool.

2.4.1.3 Brownout Management

The challenges for brownout management includes:

- When the optional services should be deactivated?

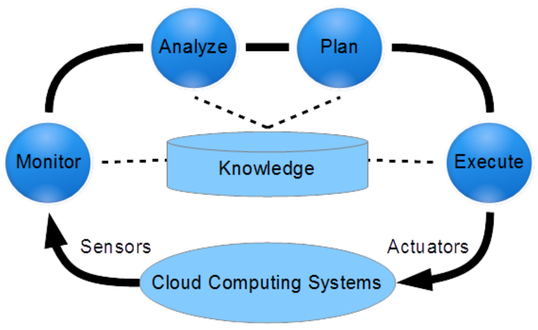


Figure 2.1: Cloud computing systems with MAPE-K adaption loop

It is relevant to system status. The optional services should be deactivated when some system indicators show the system is not running as expected. Such as, the system is becoming overloaded when requests are bursting.

- How to deactivate optional services?

  Services can be processed in different ways. For example, for the stateless services, they can be deactivated without any extra efforts. However, for the stateful services, the states should be recorded and reloaded when they are activated again.

- Which optional services should be deactivated?

  It depends on the optional service selection algorithm. The deactivated optional services can be selected based on the status of services, such as utilization.

2.4.2 Evolution of Brownout Approaches in Cloud Computing

The optimization objectives and metrics of brownout approaches in cloud computing system have been investigated and proposed throughout the years. As shown in Figure 2.2, we aim to show the evolution and development of brownout approaches in recent years.

In 2014, the application of brownout in cloud computing systems was proposed by Klein et al. [85], who introduced brownout as a self-adaptation programming paradigm. They also proposed that brownout can enhance system robustness when unpredictable requests are coming into the system. Two web application prototypes have been presented: RUBiS [4] and RuBBoS [3], which have been widely used in the follow-up research. This work showed the effectiveness of brownout approach, while the experiments

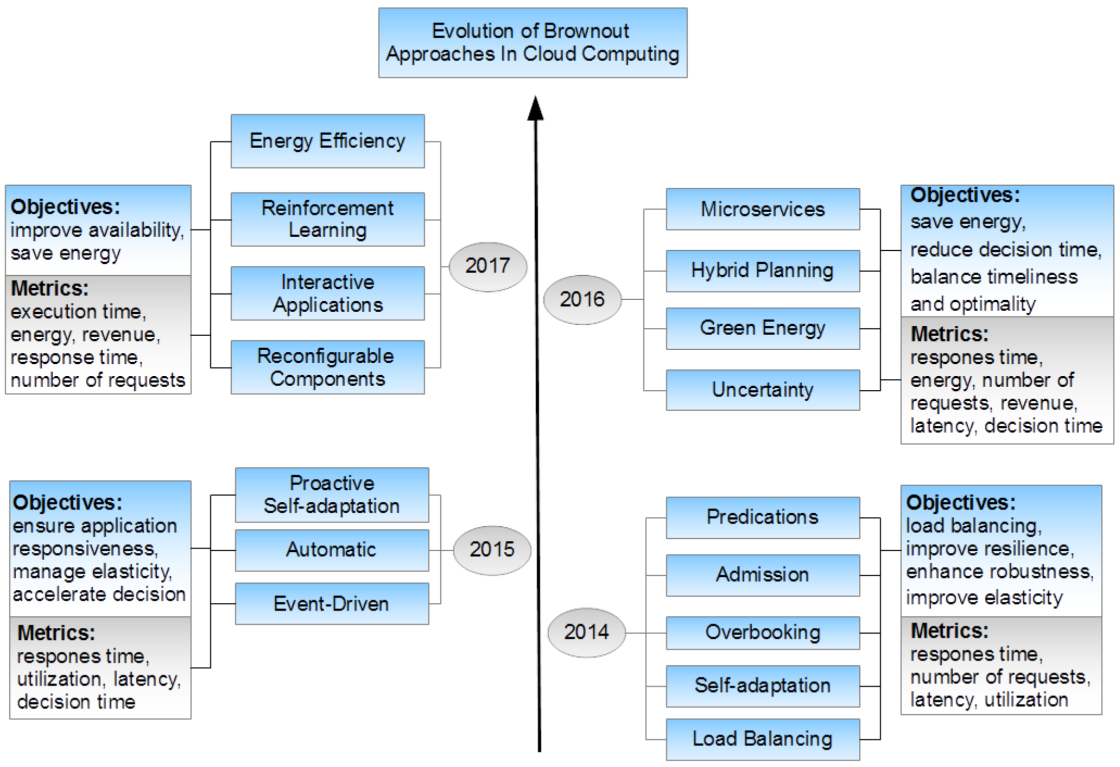

Evolution of Brownout Approaches In Cloud Computing
Energy Efficiency
Reinforcement Learning
Interactive Applications
Reconfigurable Components
2017
Objectives: improve availability, save energy
Metrics: execution time, energy, revenue, response time, number of requests
2016
Microservices
Hybrid Planning
Green Energy
Uncertainty
Objectives: save energy, reduce decision time, balance timeliness and optimality
Metrics: respones time, energy, number of requests, revenue, latency, decision time
Proactive Self-adaptation
Automatic
Event-Driven
2015
Objectives: ensure application responsiveness, manage elasticity, accelerate decision
Metrics: respones time, utilization, latency, decision time
2014
Predications
Admission
Overbooking
Self-adaptation
Load Balancing
Objectives: load balancing, improve resilience, enhance robustness, improve elasticity
Metrics: respones time, number of requests, latency, utilization

Figure 2.2: Evolution of brownout approach in cloud computing systems

were only conducted on a single machine. Durango et al. [58] presented load balancing strategies for application replicas based on brownout and admission control to maximize application performance and improve system resilience. However, the limitation is that the modeled application is not general.

To find failures earlier and avoid the limitation of periodically monitoring, based on event-driven, Klein et al. [86] proposed two load balancing algorithms for brownout-aware services. The results showed that the proposed algorithm has better performance in fault-tolerance. Maggio et al. [101] proposed and analyzed several different control strategies for brownout-aware applications to avoid overloads. Predictions for incoming load and service time are also applied in the proposed policies. To avoid service level objective (SLO) violations and unresponsive requests, Nokolov et al. [118] presented a platform to manage resource adaptively for elastic Cloud based on SLOs, which can overcome short-time request bursts. The proposed platform can adapt application execution to corresponding service level agreement (SLA). However, faults handling is not considered in this work. Tomas et al. [142] combined overbooking and brownout together to improve resource utilization without application degradation. Like [58], this approach is also based on admission control that gradually adapts application according to requests.

In 2015, automatic algorithms based on brownout had drawn more attention. Desmeurs et al. [54] presented an event-driven brownout technique to investigate the trade-offs between utilization and response time for web applications. Several automatic policies based on machine learning and admission control were also introduced in this work. This work opened a direction to establish more energy-efficient cloud data centers, while the results were not evaluated under real trace. Dupont et al. [57] proposed another automatic approach to manage cloud elasticity in both infrastructure elasticity and software elasticity. The proposed method takes advantage of the dynamic selection of different strategies. This approach considers infrastructure level and extends the application of brownout by applying it to a broader range. Moreno et al. [114] presented a proactive approach for latency-aware scheduling under uncertainty to accelerate the decision time. The motivation is applying formal model to solve the nondeterministic choices of adaptation tactics (disabling different optional contents). The limitations of this work are: 1) this work does not support concurrent tactics, and 2) the uncertainty of environment predictions is not considered.

In 2016, several articles were devoted to improving the adaption of decision time. Pandey et al. [120] proposed a hybrid selection methodology that investigates multiple optimization objectives to balance the trade-offs between different metrics, such as decision time and optimized results. Markov Decision Process (MDP) is applied to find the best choice among candidates, which represent the combination of different disabled optional contents. To overcome the limitations in [114], Moreno et al. [115] presented an approach aiming to eliminate the run-time costs when constructing MDP. The MDP is solved via stochastic dynamic programming. The results show that the decision time is reduced significantly. However, the requests are processed in an offline manner.

Some other researchers have paid their attention to energy saving for Clouds. Hasan et al. [75] introduced a green energy-aware scheduling approach for interactive cloud application that allows being dynamically adapted. Each application has three different modes, and each mode has a different percentage of optional contents.

In 2017, as an extension work of [75], Hasan et al. [74] investigated multiple metrics other than energy, including both user experience and performance for the interactive cloud application. An adaptive application management approach based on dynamically switching application modes was proposed to improve the trade-offs among multiple optimization objectives. In order to improve the elasticity of infrastructure by adding or removing resources, Hasan et al. [73] proposed a platform that applied green energy aware approach to schedule interactive applications in Clouds. The proposed platform aims to utilize both infrastructure and application resources to adapt to changing workloads.

2.5 Phases and Taxonomy of Brownout-based Adaptive Management

In this section, the phases and review of brownout approach for adaptive management of resources and applications in cloud computing systems are presented. According to our surveyed articles, we have classified adaptive management of the resources and applications with brownout into five phases: application design, workload scheduling, monitoring, brownout controller design, and metrics, as demonstrated in Figure 2.3. These phases can be mapped to the MAPE-K modules in as shown in Figure 2.1 Application design and workload scheduling correspond to the Knowledge module to describe system; monitoring is mapped to the Monitor module to monitor system status; brownout controller design corresponds to the Analyze, Plan and Execute modules; and metrics are mapped to the information obtained from the Sensors. Now, we explain the details of each phase.

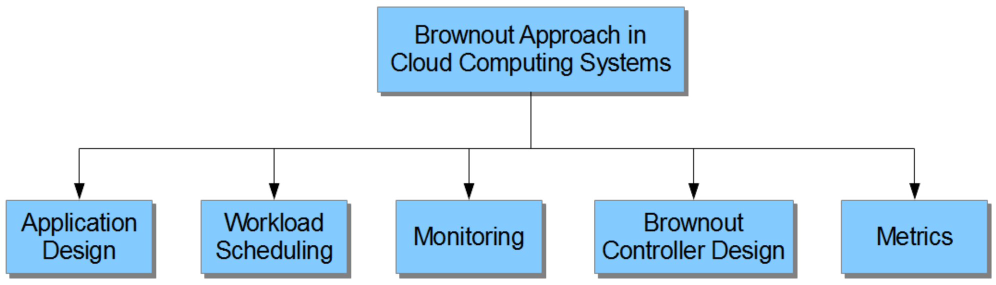

Brownout Approach in Cloud Computing Systems
Application Design
Workload Scheduling
Monitoring
Brownout Controller Design
Metrics

Figure 2.3: Phases of applying brownout approach in cloud computing systems

2.5.1 Application Design

Applications are running in cloud computing systems and providing services for users. To enhance the system performance in Clouds, applications are required to be designed by referring to system configuration and users requirements. In Figure 2.4, the categories we used for the application design are: 1) application type, 2) application domain, 3) optional parts, and 4) application deployment.

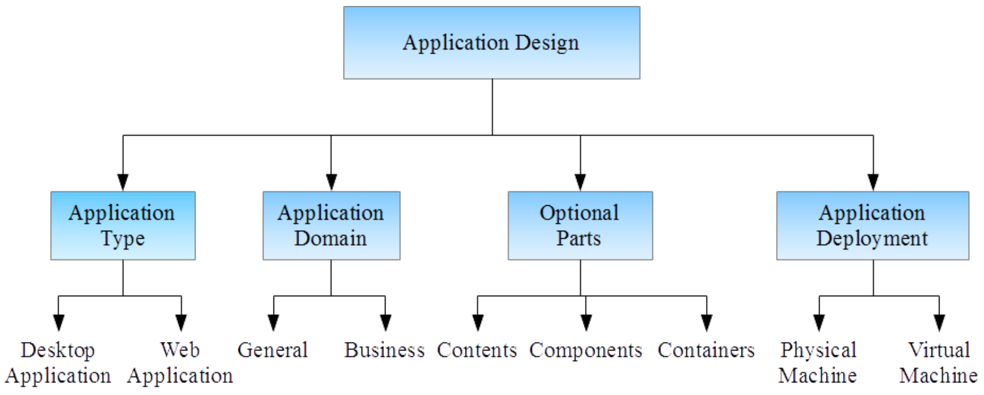


Figure 2.4: Taxonomy based on application design

5.1.1 Application Type. The application can be running locally or in the remote manner enabled by brownout. Brownout-enabled applications can be classified into two types according to application type: (1) desktop application, and (2) web application. The desktop application represents the brownout-enabled applications runs locally on machines, for example, the texts processing application [21]. The web application is implemented in the client-server model to provide web services that the users interact through the Internet. The typical brownout-enabled web application is online shopping system, which has been applied in many existing brownout-related articles [58][85][101].

5.1.2 Application Domain. Applications are implemented to provide functionalities for users. The developer is aiming to develop applications in a more efficient manner, while the complexity of applications is growing. Adaptive application management in cloud computing systems provides an approach to managing complex applications. To manage these applications, the domain of applications should be identified, as applications in different domains have different management requirements. For the applications in the general domain, applications are providing functions for general purposes, such as scientific calculation applications. While for the applications in the business domain, which focuses on maximizing the profits of service providers, they are more sensitive to some specific performance metrics. For example, in the online shopping system that belongs to the business domain, the response time as QoS requirement is one of the most critical metrics [74], since long response time or unresponsiveness leads to the loss of users.

5.1.3 Optional Parts. In brownout-enabled applications, the optional parts in the applications are temporarily deactivated to manage resources and applications. The optional parts represent the scheduling units in the applications. In existing articles, the optional parts are identified as (1) contents, (2) components and (3) containers. Optional web contents on servers are to be showed selectively to users to save resource usage [85]. Components-based applications deactivate optional components to manage resource utilization [21]. In containerized clouds, each service is implemented as containers, and the optional containers can be activated/deactivated based on system status.

5.1.4 Application Deployment. In cloud computing systems, applications can be deployed on physical machines (PMs) or virtual machines (VMs). Deploying applications directly on PMs enables applications to have almost the same performance as native systems, such as containerized applications. One PM can host multiple VMs, and multiple applications can be deployed on the same VM to improve the usage of shared resources. When applications are deployed on VMs, VM migrations and VM consolidation can be applied with brownout together to optimize resource utilization.

2.5.2 Workload Scheduling

Workload scheduling aims at scheduling and allocating appropriate resources to suitable workloads according to SLA defined by end-users and service providers so that the workloads can be executed efficiently and the applications utilize resources efficiently. In Figure 2.5, the categories we used for workload scheduling are: 1) workload type, 2) resource type, 3) dynamicity, 4) workload trace and 5) experiment platform.

2.5.2.1 Workload Type. The workload type represents the resource requirement of workloads. In current brownout-relevant articles, most works are focusing on scheduling the CPU-intensive workloads [58][54][57], as computation resources are regarded as the main resource allocated to workloads. Some articles also consider network-intensive workloads to reduce the network latency [118].

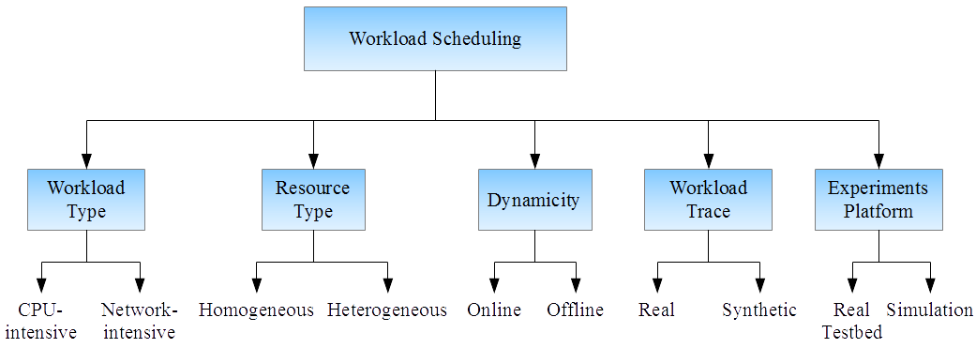


Figure 2.5: Taxonomy based on workload scheduling

2.5.2.2 Resource Type. For homogeneous resource type, the resources offered by service providers are limited to a single type. This configuration simplifies the workload scheduling and overlooks the various characteristic of workload. The homogeneous configuration is mostly used in small-scale tests where resource diversity is limited or for the initial research of novel approach [58][85]. Cloud computing systems have the nature of heterogeneity, to utilize this feature, mature service providers are offering heterogeneous resources for workload scheduling. For example, Amazon EC2 has provided more than 50 types of VMs, and these VMs are categorized as various classifications for different workload types, such as general purpose, computation intensive purpose or memory intensive purpose [59].

2.5.2.3 Dynamicity. The dynamicity of workload scheduling represents the time when the workload information is obtained. The dynamicity of workload scheduling is noted as online if the information of workloads is only available when the workloads are coming into systems. If all the information of workloads are known in advance and workloads can be rescheduled based on system resource, this scheduling process is identified as offline. One example of the offline scheduling is the batch job [75], in which the deadlines of jobs are known and jobs can be executed with delay based on resource availability. Compared with online workload scheduling, offline workload scheduling is prone to achieve more optimized results, however, it is not available for some scheduling scenarios, such as real-time applications.

2.5.2.4 Workload Trace. Different workload traces are used to evaluate the performance of brownout approaches. When brownout was initially proposed, synthetic traces, such as workloads generated based on Poisson distribution, were applied. Later on, workloads derived from real traces were also applied. Presently, the popular real traces are from FIFA [5], Wikipedia [151], and Planet lab trace [122].

2.5.2.5 Experiments Platform. Both real testbed and simulations have been conducted to test the performance of brownout approaches. Experiments under real testbed are more persuasive. Grid'5000 platform [67] has been adopted in several articles, which provides APIs to collect PM utilization and energy consumption. However, some uncontrolled factors exist in real testbed, such as network traffics and unpredictable loads. Therefore, simulation tools provide more feasible options. Besides, with simulations, it is easier to conduct experiments with heterogeneous resources as well as large-scale size. The cloud simulation toolkit, CloudSim [40], has been used for simulating brownout-enabled workload scheduling.

2.5.3 Monitoring

The objective of monitoring is achieving performance optimization by monitoring resource usage in cloud computing systems. Therefore, a monitoring component is required to collect system and analyze the resource utilization information. After analysis, decisions to change the system status and resource usage are made to ensure system to satisfy the specified SLA. As shown in Figure 2.6, we categorize the components of monitoring as 1) resource usage, 2) services, 3) status, and 4) execution.

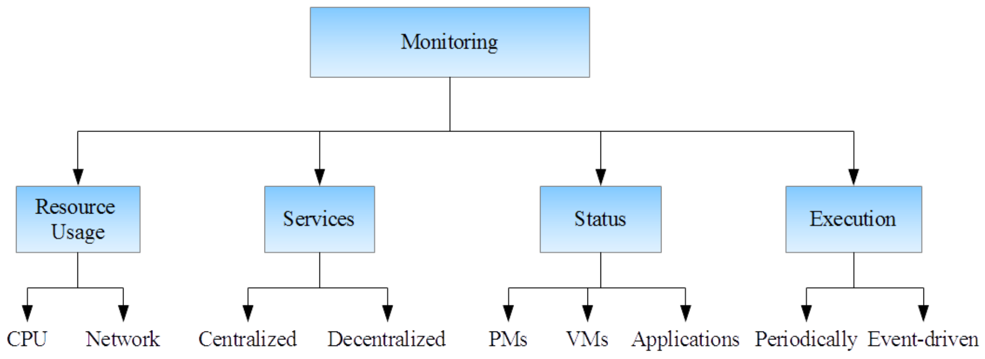


Figure 2.6: Taxonomy based on monitoring

2.5.3.1 Resource Usage. The monitor in Clouds environment is used to monitor resource usage, including CPU, memory and network usage via the monitoring tools. As mentioned in Section 2.5.2.1, the current brownout related workload scheduling is focusing on handling CPU-intensive and network-intensive workloads [86][118]. So, the monitors in brownout-enabled systems are mainly monitoring the CPU and network resource usage. Two objectives of monitoring resource usage can be achieved from both users' and service providers' perspectives: users expect their requests to be processed within QoS, and service providers aim to execute the workloads with minimum resource usage.

2.5.3.2 Services. Service monitoring gathers the information about resource statuses to check whether the workload is executed by applications as desired. Two types of service monitoring exist in cloud computing systems, one is centralized and the other is distributed. In a centralized manner, a central repository is applied to store the collected data, which is not scalable when the number of monitored targets is increased. For the distributed service monitoring, the monitored data is stored distributedly to achieve better fault tolerance and load balancing [21].

5.3.3 Status. To be more specific, monitoring resource utilization is regarded as monitoring the status of different levels, including PMs [114], VMs [57], and applications [142]. Based on various optimization goals, data are collected at different levels. For example, to reduce the energy of cloud computing systems, power consumption of PMs should be collected. To improve the response time of requests, it is required to obtain the resource usage of applications.

2.5.3.4 Execution. To avoid unexpected failures and execute workloads promptly, the execution monitoring has two types: periodically and event-driven. In the periodical manner, the monitor periodically checks the resource usage and makes decisions on the execution process for the next time period [58]. In an event-driven manner, changes in the execution process are triggered once a specific event is detected, such as resource usage is above the predefined overloaded threshold [86]. The motivation of event-driven is its real-time requirement, which is suitable for latency-aware applications.

2.5.4 Brownout Controller/Dimmer Design

In brownout-enabled systems, the deactivation/activation operations on optional parts are managed by brownout controller. The brownout controller also has a control knob called dimmer, which represents the probability of the optional parts to be deactivated. Therefore, we can notice that the brownout controller design is the most important part for brownout approach. In Figure 2.7, we have categorized the classification of brownout controller/dimmer design as 1) parameters, 2) controller algorithm, 3) controller number and 4) adaptivity.

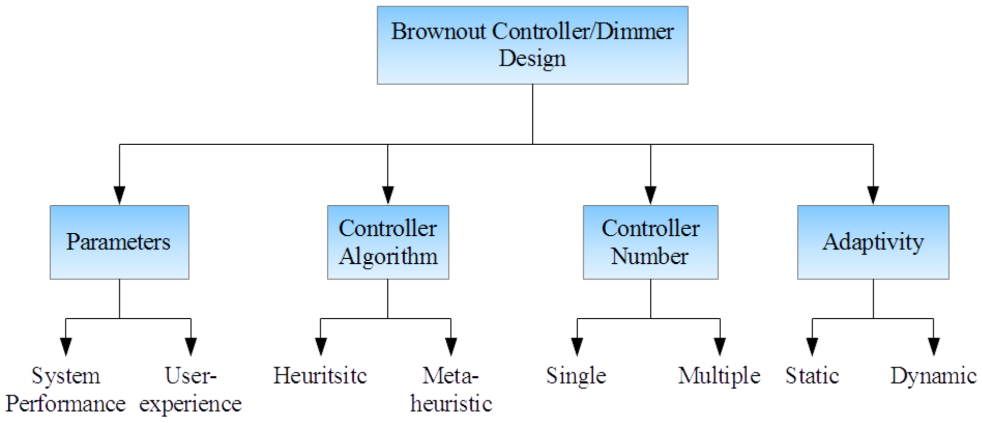


Figure 2.7: Taxonomy based on brownout controller design

2.5.4.1 Parameters. Brownout controller can be designed based on different parameters. These parameters can be classified based on system and user perspectives as system performance and user experience. If system performance is addressed, the brownout controller is configured with system parameters, such as resource utilization [58]. If the brownout controller aims to optimize user-experience, parameters like response time can be designed into brownout controller [85].

2.5.4.2 Controller Algorithm. Similar to the resource scheduling problem, finding the optimal solutions of suitable optional parts to be deactivated/activated by controller algorithms is computationally expensive. Thus, finding the approximate solutions is an alternative to the most of proposed approaches. We have classified the surveyed controller algorithms as heuristic and meta-heuristic. The heuristic ones comply with the predefined constraints and try to find an acceptable solution for particular problem [137]. Usually, the constraints are varied for different problems, and the solutions can be obtained within a limited time. One type of heuristic algorithms is greedy algorithm and has been adopted in [54][57][114]. As for meta-heuristic algorithms, they are generally applied to general purpose problems [137], and have standard procedures for problem construction and solving. One example of the meta-heuristic algorithm is the approach based on Markov Decision Process that has been applied in [115].

2.5.4.3 Controller Number. The brownout controller may have single or multiple controllers to manage the optional parts in cloud computing systems. In [142], multiple controllers are applied, in each application, there is a controller and its dimmer value is calculated based on its status. Thus, the dimmer values of different applications in [142] can be varied. For overall management, a single controller is applied. For example, in [73], the dimmer value is calculated as the severity of overloads in the whole data center.

2.5.4.4 Adaptivity. The adaptivity represents whether the brownout controller is adaptive to the change of workloads. It is categorized as static and dynamic. In our surveyed approaches, most of them are dynamic [58][101][142], which can be dynamically adapted to the change of system. Only limited approaches are static, which apply static parameters. The static brownout controllers are easy to violate the specific SLAs [75][73].

2.5.5 Metrics

Different metrics are considered in cloud computing systems to evaluate the performance of different brownout-based approaches. As shown in Figure 2.8, from our surveyed literature, we have identified 9 metrics: response time, execution time, utilization, availability, decision time, latency, request number, energy and revenue.

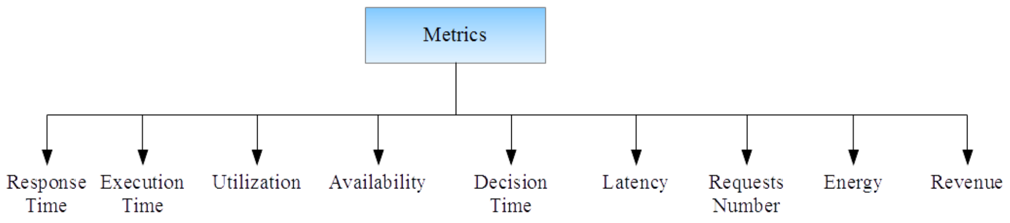


Figure 2.8: Taxonomy based on metrics

Response Time [58][85][101] is the total time it costs from when users send requests until they receive a response. The response time can be influenced by system processing time, which is also relevant to hardware resource utilization. This metric should be reduced to improve the user experience. *Execution Time* [21][154] is the time required to complete the workloads execution. Minimizing the execution time can improve system

QoS. *Utilization* [118][142] is the actual resource percentage used to run workloads to the total resource provided by service provider. Available utilization should be improved to maximize the resource usage. *Availability* [21][154] is the capability of the cloud computing system to guarantee the services are available with expected performance as well as handle fatal situations. This metric should be ensured in cloud computing systems, e.g. during 99.9% time, the services are running with expected performance. *Decision Time* [114][115] is the time that scheduling algorithm takes to find the optional parts to be deactivated/activated, which is associated with algorithm design. To find the solutions faster, in online scheduling scenario, the decision time of algorithm should be accelerated. *Latency* [85][114] is the time delay that spends on message communication across the network, which is relevant to hardware resources and utilization. In business applications, latency is an crucial metric to indicate the system performance. *Requests Number* [57][75] is the number of requests received by the system. The scheduling algorithm has better performance if more requests can be served with the same amount of resources. *Energy* [73] is the total power used for the cloud computing system to provide the services to users. The system should reduce energy consumption while ensuring QoS. *Revenue* [73][75] is the profit obtained from users when service providers are offering services. The service provider aims to maximize their revenue as well as lowering costs.

2.6 Review of Brownout Approaches

In this section, a review of brownout-based approaches for adaptive management of resources and applications in cloud computing system is conducted. To identify the differences in surveyed articles, we use the taxonomy in Section 2.5 to map the key features of these approaches. Table 2.2 shows a summary of selected brownout approaches, and Tables 2.3 to 2.7 summarize the comparison of selected brownout approaches and their categorized classification according to our taxonomy. For instance, Table 2.3 shows the comparison based on the taxonomy of application design as presented in Section 2.5.1, and Table 2.7 shows the metrics comparison based on the discussion in Section 2.5.5.

The "Convex Optimization Based Load Balancing" (COBLB) [58] technique extends brownout approach for services with multiple replicas, which are the copies of applica-

Table 2.2: Summary of brownout approaches

Approach	Year	Author	Description	Orgranization
COBLB [58]	2014	Durango et al.	Load Balancing with Brownout and Control Theory	Umea University, Sweden
SAB [85]	2014	Klein et al.	Robust Cloud Applications with Brownout	Umea University, Sweden
EPBH [86]	2014	Klein et al.	Resilience and Load Balancing with Brownout	Umea University, Sweden
FPF [101]	2014	Maggio et al.	Brownout Prediction	Lund University, Sweden
CLOUDFARM [118]	2014	Nikolov et al.	Adaptive Resource Management	University of Ulm, Germany
BOB [142]	2014	Tomas et al.	Overbooking with Brownout	Umea University, Sweden
EDB [54]	2015	Desmeurs et al.	Event-Driven Application with Brownout	Umea University, Sweden
CAA [57]	2015	Dupont et al.	Cloud Elasticity with Brownout	INRIA, France
PLA [114]	2015	Moreno et al.	Proactive Self-Adaptation for Uncertainty Environment	Carnegie Mellon University, USA
HYB-Q [75]	2016	Hasan et al.	Green Energy for Cloud Application	INRIA, France
PLA-MDP [115]	2016	Moreno et al.	Decision-Making for Proactive Self-Adaptation	Carnegie Mellon University, USA
HYBP [120]	2016	Pandey et al.	Hybrid Planning in Self-Adaptation	Carnegie Mellon University, USA
MODULAR [21]	2017	Alvares et al.	Modular Autonomic Manager in Software Components	INRIA, France
SaaScaler [74]	2017	Hasan et al.	Power and Performance Scaler for Applications	INRIA, France
GPaaScaler [73]	2017	Hasan et al.	Green Energy Aware Scaler for Interactive Applications	INRIA, France
RLBF [154]	2017	Zhao et al.	Reinforcement Learning for Software Adaptation	Peking University, China

Table 2.3: Taxonomy based on application design

Approach	Application Type	Application Domain	Optional Parts	Application Deployment
COBLB	Web application	Business	Contents	Virtual Machine
SAB	Web application	Business	Contents	Virtual Machine
EPBH	Web application	Business	Contents	Virtual Machine
FPF	Web application	Business	Components	Physical Machine
CLOUDFARM	Web application	Business	Components	Physical Machine
BOB	Web application	Business	Contents	Virtual Machine
EDB	Web application	Business	Contents	Virtual Machine
CAA	Web application	Business	Contents	Virtual Machine
PLA	Web application	Business	Contents	Virtual Machine
HYB-Q	Web application	Business	Contents	Virtual Machine
PLA-MDP	Web application	Business	Contents	Virtual Machine
HYBP	Web Application	Business	Contents	Virtual Machine
MODULAR	Desktop Application	General	Components	Physical Machine
SaaScaler	Web application	Business	Contents	Virtual Machine
GPaaScaler	Web application	Business	Contents	Virtual Machine
RLBF	Web application	Business	Components	Physical Machine

Table 2.4: Taxonomy based on workload scheduling

Approach	**Workload Type**	**Resource Type**	**Dynamicity**	**Trace**	**Experiments Platform**
COBLB	CPU-intensive	Homogeneous	Online	Synthetic	Simulation
SAB	CPU-intensive	Homogeneous	Online	Synthetic	Real testbed
EPBH	CPU-intensive	Homogeneous	Online	Synthetic	Real testbed
FPF	CPU-intensive	Homogeneous	Online	Synthetic	Simulation
CLOUDFARM	Network-intensive	Heterogeneous	Online	Synthetic	Real testbed
BOB	CPU-intensive	Homogeneous	Online/Offline	Real	Real testbed
EDB	CPU-intensive	Homogeneous	Online	Synthetic	Real testbed
CAA	CPU-intensive	Homogeneous	Offline	Synthetic	Real testbed
PLA	CPU-intensive	Homogeneous	Online	Real	Real testbed
HYB-Q	CPU-intensive	Homogeneous	Online	Real	Real testbed
PLA-MDP	CPU-intensive	Homogeneous	Offline	Real	Real testbed
HYBP	CPU-intensive	Heterogeneous	Online	Real	Simulation
MODULAR	CPU-intensive	Homogeneous	Online	Synthetic	Real testbed
SaaScaler	CPU-intensive	Homogeneous	Online	Real	Real testbed
GPaaScaler	CPU-intensive	Homogeneous	Online	Real	Real testbed
RLBF	CPU-intensive	Homogeneous	Online/Offline	Synthetic	Real testbed

Table 2.5: Taxonomy based on monitoring

Approach	**Resource Usage**	**Services**	**Status**	**Execution**
COBLB	CPU	Centralized	Applications	Periodically
SAB	CPU	Centralized	Applications	Periodically
EPBH	CPU	Centralized	Applications	Event-Driven
FPF	CPU	Centralized	Applications	Periodically
CLOUDFARM	Network	Centralized	PMs, Applications	Periodically
BOB	CPU	Centralized	Applications	Periodically
EDB	CPU	Centralized	Application	Event-Driven
CAA	CPU	Centralized	VMs	Periodically
PLA	CPU	Centralized	PMs	Periodically
HYB-Q	CPU	Centralized	Applications	Periodically/Event-Driven
PLA-MDP	CPU	Centralized	PMs	Periodically
HYBP	CPU	Centralized	PMs	Periodically
MODULAR	CPU	Decentralized	PMs	Periodically/Event-Driven
SaaScaler	CPU	Centralized	VMs, Applications	Periodically/Event-Driven
GPaaScaler	CPU	Centralized	VMs, Applications	Periodically
RLBF	CPU	Centralized	PMs	Periodically

Table 2.6: Taxonomy based on brownout controller/dimmer design

Approach	**Parameters**	**Controller Algorithm**	**Controller Number**	**Adaptivity**
COBLB	System Performance	Heuristic	Multiple	Dynamic
SAB	User-experience	Heuristic	Multiple	Dynamic
EPBH	System Performance	Heuristic	Single	Dynamic
FPF	User-experience	Heuristic	Single	Dynamic
CLOUDFARM	System Performance	Heuristic	Multiple	Dynamic
BOB	System Performance	Heuristic	Multiple	Static/Dynamic
EDB	System Performance	Heuristic	Multiple	Dynamic
CAA	System Performance	Heuristic	Multiple	Dynamic
PLA	System Performance	Meta-heuristic	Single	Dynamic
HYB-Q	User-experience	Heuristic	Single	Static
PLA-MDP	System Performance	Meta-heuristic	Single	Static
HYBP	System Performance	Meta-heuristic	Single	Dynamic
MODULAR	System Performance	Heuristic	Multiple	Dynamic
SaaScaler	User-experience	Heuristic	Single	Static
GPaaScaler	User-experience	Heuristic	Single	Static
RLBF	System Performance	Meta-heuristic	Multiple	Static/Dynamic

Table 2.7: Taxonomy based on metrics

Approach	**Response Time**	**Execution Time**	**Utilization**	**Availability**	**Decision Time**	**Latency**	**Requests Number**	**Energy**	**Revenue**
COBLB	✓						✓		
SAB						✓	✓		
EPBH	✓						✓		
FPF	✓						✓		
CLOUDFARM			✓						
BOB	✓		✓						
EDB	✓		✓						
CAA	✓						✓		
PLA					✓	✓			
HYB-Q	✓						✓	✓	✓
PLA-MDP					✓	✓			
HYBP	✓				✓				
MODULAR		✓		✓					
SaaScaler	✓						✓	✓	✓
GPaaScaler	✓							✓	✓
RLBF		✓		✓					

Table 2.8: Comparison of brownout approaches

Approach	Objectives	Advantages	Limitations
COBLB	Load balancing for Cloud applications	Resilience is improved	Application model is not general
SAB	Enhance robustness of Cloud applications	Support more users with fewer resources	Only tested on a single machine
EPBH	Improve resilience	Response time is reduced	Parameters are chosen empirically
FPF	Mitigate overloads	Prediction is applied	Only validated with simulations
CLOUDFARM	Improve elasticity	SLA is ensured when there are bursts	Faults handling is not considered
BOB	Resource overbooking	Improve resource utilization without application degradation	Scalability is not discusses
EDB	Balance utilization and response time	Utilization is improved and response time is ensured	Not evaluated with real workloads
CAA	Manage elasticty	Elasticity is improved at both infrastructure and software levels	Limited number of tactics
PLA	Accelerate decision time	Latency is reduced	Concurrent tactics are not supported
HYB-Q	Green energy provisioning for interactive cloud application	Brown energy is saved	Non-interactive cloud applications are not investigated
PLA-MDP	Accelerate decision time	Decision time is reduced	Not available for online requests
HYBP	Balance decision time and optimality	Decision time is reduced	Not validated in real testbed
MODULAR	Manage modular components	System availability is improved	The availability for other applications is not discussed
SaaScaler	Investigate trade-offs between energy and performance	Energy is reduced and QoS is improved	Resources addition or removal are not flexible
GPaaScaler	Reduce energy consumption	Energy is saved	Not available for real-time requests
RLBF	Enhance flexibility of approach	Better adaptation effectiveness is achieved	Convergence time is not analyzed

tions providing same functionalities. These replicas contain optional contents and allow to be served selectively according to system status. To enhance the system load balancing performance, the technique also collects information about adaptation in replicas. In the technique, all replicas are managed in queuing systems adopting the resource sharing. Response time and the number of requests are improved by this technique.

Self-Adaptive Brownout (SAB) [85] approach adds a dynamic parameter that affects user experience and the amount of resource required by the application. This parameter is adapted based on both workload and available resources to enhance the robustness of applications. The proposed approach is synthesized with control theory approach to adaptively determine when to activate/deactivate optional features in the applications. With control theory, the behaviors of applications can be predicted, and some system constraints can be guaranteed. In this approach, the maximum latency is controlled so that the latency is reduced. Also, SAB can serve more users with fewer resources.

Equality Principle-Based Heuristic (EPBH) [86] is focusing on enhancing the resilience of cloud services when system faces resource shortage. Similar to COBLB [58], this approach is also applied to application replicas. This event-driven approach is based on heuristic and requires all the replicas to have a control parameter (dimmer) value. According to the parameter value, EPBH decides which replicas to receive more loads. EPBH has been proven to improve resilience when resource shortage is triggered by failures.

The objective of Feedforward Plus Feedback (FPF) [101] approach is keeping the average response time below a certain threshold. FPF works by configuring the probability to serve requests with optional computations. FPF is able to handle the requests bursts for cloud applications and reacts to the changes of resource amount allocated to cloud applications. The idea of FPF is obtaining the number of currently queued requests from the applications and predicting the latency experienced by future requests. The results demonstrate that although FPF spends efforts on obtaining more information, the overloads are mitigated.

CLOUDFARM [118] is an elastic cloud platform to manage resource reservations in flexible and adaptive ways based on actual resource demands and predefined SLAs. It provides flexible SLAs that can be configured dynamically to fulfill the elasticity of cloud computing systems. Based on SLAs and costs produced by users, the services can be dynamically downgraded to avoid bursts so that system utilization and revenue can be improved. CLOUDFARM also introduces an abstract level by using virtual framework for all the applications, which benefits from its lightweight and dynamic degradation from full mode to degraded mode.

Brownout OverBooking (BOB) [142] technique is designed to ensure graceful degradation when loads spike and thus avoiding overloads in resource overbooking scenario. The motivation of BOB is combining overbooking and brownout together, where overbooking system takes advantage of application information from brownout and applies deactivation operations to relieve overloads. In BOB, an overbooking controller is responsible for admitting or rejecting new requests and a brownout controller is responsible for adapting the resources allocated to accepted requests. Higher utilization is achieved while response time is ensured by BOB.

To ensure application responsiveness, an Event-Driven Brownout (EDB) [53] technique at application level has been proposed. In this approach, the application is allowed to run optional codes that are not necessary for key functionalities but for extra user experience. EDB combines machine learning techniques and control theory together to dynamically configure a given threshold that represents the number of pending requests. The configuration operation is based on application response time. An advance in ensuring response time is achieved without sacrificing utilization.

Cloud Automatic Approach (CAA) [57] aims at managing cloud elasticity in a cross-layered manner. The motivation is combining the infrastructure elasticity and software elasticity to overcome the conceptual limitations of IaaS and make software at SaaS involved in the elasticity process. For the software at SaaS, they are running at different levels and can be degraded to lower levels when the resource is limited. And for the resources at IaaS level, physical machines can be scaled in and out according to system loads. An advantage in response time is obtained when the cross-layered manner is working in a coordinated way.

Proactive Latency-Aware (PLA) [114] addresses the inefficient reactive adaption problem when adaptation has latency. Thus, PLA considers adaptation latency and applies the probabilistic model to decide adaptations. The main motivation is using a formal model to find the adaptation decisions which are underspecified through nondeterminism, and then resolve the nondeterministic choices to maximize optimization goals. PLA takes the uncertainty of environment into account and predicts needed resources by looking ahead. The effectiveness of adaptation decisions is significantly improved.

The QoS aware hybrid controller (HYB-Q) [75] technique is aiming at achieving green energy-aware scheduling by using both green and brown energy. The target applications are focused on interactive cloud applications that can be dynamically adapted to available green energy based on changing conditions. HYB-Q obtains information in feedback loop about the number of requests executed under different modes in the previous time period and computes the current adaptation value. In the controller, the response time and workload changes are periodically checked. If the response time arises above the predefined threshold, the user experience is downgraded to lower mode to avoid overloads by the controller. It is observed that providers' revenue is improved and the usage of brown energy is reduced.

As an extension work of PLA [114], Proactive Latency-Aware with Markov Decision Process (PLA-MDP) [115] takes advantage of the probabilistic property of Markov Decision Process to adapt application functionalities. To eliminate the run-time overhead of constructing MDP, stochastic dynamic programming is applied to solve MDP. The decision time is vitally reduced while same results are obtained.

The objective of Hybrid Planning (HYBP) [120] is finding the trade-offs between timeliness and optimality of solutions to adapt application modes. HYBP combines two approaches together to achieve the best balance between the conflicts. One is deterministic planning and the other is Markov Decision Process planning, which can be fitted in different cases. In the case of fast response is required, the deterministic planning is applied to give a fast response. When the time is adequate, the MDP planning is applied to generate an optimal solution. Simulated experiments have shown that HYBP can improve system performance by balancing the aforementioned trade-offs.

MODULAR [21] approach leverages modularity feature of the component-based system to strengthen the domain-specific language application. Domain-specific language has high-level constructs to describe the configurations and executed policies of the target system. With brownout mechanism, the components can be transferred from nominal configuration to degraded one, which incurs different loads for the system. Thus, the system is entitled the capability of dynamic reconfiguration to be self-adaptive.

Compared with HYB-Q [75] that focuses on green energy usage, SaaScaler [74] approach analyzes the trade-off between power and performance by considering green energy usage for the interactive cloud applications. SaaScaler can satisfy different metrics. Considering these metrics, three levels of user experience are defined. Capacity requirement is dynamically adjusted among these levels to serve more users. Experiments conducted in real testbed show that energy consumption is reduced while performance and revenues are improved by carefully tuning applications.

GPaaScaler [73] considers adaptation both at infrastructure and application levels by using green energy source. At the infrastructure level, resources are added or removed according to resource demand of applications. While the applications are dynamically changing their service levels according to performance and green energy availability. These adaptations are implemented separately and can be coordinated together. Lower costs and less usage of brown energy are achieved.

The objective of Reinforcement Learning-Based Framework (RLBF) [154] approach is overcoming the limitations of rule-based adaptation that decisions are only made based on static rules. Based on reinforcement learning, RLBF enables automatic learning rules with various optimization objectives in an offline manner. Additionally, the rules can also be evolved with real-time information for online adaptation of selecting optional services to run. The efficiency and effectiveness of adaptation process are improved by this approach.

To summarize the merits and demerits of reviewed brownout approaches, a comparison of the advantages and limitations of each brownout approach is presented in Table 2.8. For example, COBLB [58] approach achieves better resilience while the adopted application model is not general.

2.7 A Perspective Model

Brownout approach has shown its ability to improve applications and resources management to handle changes in workloads. However, the trade-offs between workload

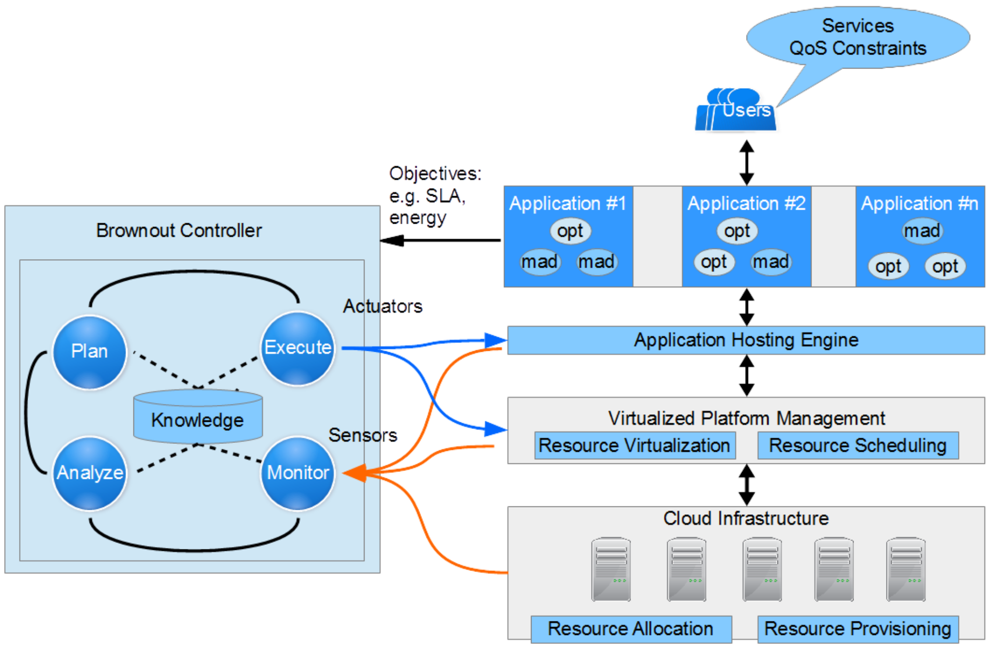


Figure 2.9: Perspective model of applying brownout in cloud computing systems

handling and performance degradation are necessary for consideration. Several optimization objectives are usually considered together to investigate the trade-offs. The primary research problems in the context are as below:

- (1). How to enable brownout approach in cloud computing system?
- (2). How to enable brownout to manage resources and applications adaptively?
- (3). How to balance the trade-offs between different metrics when applying brownout?

To deal with these issues, there is a need for the brownout-enabled mechanism for adaptive management. We propose a perspective model of brownout-enabled cloud computing system for adaptive resource scheduling as shown in Figure 2.9.

From users' perspective, users interact with the system and submit their requests for services to the system. The users may have constraints for the submitted requests, such as QoS constraints or budget. From service providers' perspective, these workloads generated by users are executed by applications.

Applications are provided by service providers to offer services for users, and these applications are managed by the application hosting engine, e.g. Docker [11] and Apache

Tomcat [14]. Applications can be deployed on either virtualized platform (virtual resources) or cloud infrastructure (physical resources). The host application engine can be container-based management platforms, e.g. Docker Swarm [17], Kubernetes [16] or Mesos [13], which provide management of container-based applications. The virtualized platform manages the virtualized resources, for example, the Virtual Machines managed by VMware [18]. As for the resource allocation and provisioning in cloud infrastructure, they can be managed by infrastructure management platform, such as OpenStack [15].

To deal with research problem (1), the applications can be composed of optional and mandatory components/services. The optional components/services should be identified by service providers and can be activated/deactivated by brownout controller. For instance, in Docker Swarm, the services can be deployed via a configuration file [10] and the file can set the service with the feature ”optional”.

To resolve research problem (2), a brownout controller is required based on MAPE-K architecture model and fits into the feedback loop of MAPE-K model as introduced in Figure 2.1. As described in Section 2.4.1.2, it has modules including Monitor, Analyze, Plan and Execute to fulfill adaptation with cloud computing system. Sensors and Actuators are used to interact with cloud computing system. Sensors collect the information from different levels in cloud computing systems, including application hosting engine, virtualized platform, and cloud infrastructure. The Sensors can be the devices attached to hardware, e.g. power meter. The collected information is provided to Monitor module.

After analyzing the information by Analyze module, Plan module makes decisions for applying brownout control operations, in which the brownout-based scheduling policies are implemented. Based on the decisions, the Execute module applies brownout via actuators to application hosting engine and virtualized platform to enable/disable optional components/services. These operations can be fulfilled via the APIs provided by application hosting engine or virtualized platform.

To handle the research problem (3), the Knowledge pool in MAPE-K model is applied to store the predefined objectives (e.g. load balancing, energy efficiency or SLA constraints) and trade-offs (e.g. trade-offs between energy and SLA). The rules in Knowledge pool, such as SLA rules, can be updated according to brownout-based policies.

2.8 Summary

Brownout is a novel paradigm for self-adaptation that enables optional parts in the system to be temporarily disabled in order to deal with changing situations. In addition, it has been shown as a promising approach to improve system performance and user experience.

In this chapter, we proposed a review and taxonomy of brownout-based adaptive resources and applications management for cloud computing systems. A taxonomy is presented according to 5 phases: (1) application design, (2) workload scheduling, (3) monitoring, (4) brownout controller/dimmer design and (5) metrics. The taxonomy of each phase is summarized based on the different classification of brownout approaches. Then, a comprehensive review of existing brownout approaches was presented. Furthermore, the comparison of the advantages and limitations of the surveyed brownout approaches are also made.

Our analysis shows that brownout can be applied in cloud computing systems for different optimization objectives. Brownout approach also provides a new option for adaptive management of resources and applications. This chapter shows the progress of brownout since it was firstly borrowed to Clouds, and it also helps readers to find the research gap existing in the discussed area. It is a promising direction to combine brownout with other existing techniques to achieve better system performance. In the following chapters, we present our research contributions in this area.

Chapter 3

Energy Efficient Scheduling of Cloud Application Components with Brownout

Dynamic Voltage Frequency Scaling and VM consolidation have been proved effective to manage overloads. However, they cannot function when the whole data center is overloaded. Brownout provides a promising direction to avoid overloads through configuring applications to temporarily degrade user experience. Additionally, brownout can also be applied to reduce data center energy consumption. In this chapter, as a complementary option for Dynamic Voltage Frequency Scaling and VM consolidation, we propose a brownout-based approach to reduce energy consumption through selectively and dynamically deactivating application optional components, which can also be applied to self-contained microservices.

3.1 Introduction

GIVEN the scenario that the budget and resource are limited, overloaded tasks may trigger performance degradation and lead the applications to saturate, in which some applications cannot be allocated by provider. Therefore, some users are not served in a timely manner or experience high latencies, others may even not receive service at all [85]. The saturated applications also bring the over-utilized situation to hosts and cause high energy consumption. Unfortunately, current resource management approaches like Dynamic Voltage Frequency Scaling (DVFS) and VM consolidation cannot function when the holistic data center is overloaded.

This chapter is derived from:
• **Minxian Xu**, Amir Vahid Dastjerdi, and Rajkumar Buyya, "Energy Efficient Scheduling of Cloud Application Components with Brownout," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 1, Number 2, Pages: 40-53, ISSN: 2377-3782, IEEE Computer Society Press, USA, July-Dec 2016.

In this chapter, we consider component-level control in our system model. The model could also be applied to the container or microservices architecture. We model the application components as mandatory and optional, if required, optional components can be deactivated. By deactivating the optional components selectively and dynamically, the application utilization is reduced and eventually total energy consumption is saved as well. While under the market scenario, the service provider may provide discount to user as the services are deactivated.

Our objective is to tackle the problem of energy efficiency and our contributions are as below:

- Our approach considers the trade-offs between discount that should be given to a user if a component is deactivated and how much energy can be saved.
- Then we propose a number of policies that consider the aforementioned trade-offs and dynamically make decisions on which components are going to be deactivated.

The rest of this chapter is organized as: after discussing related work in Section 3.2, we present the brownout enabled system model and problem statement in Section 3.3. Section 3.4 introduces our proposed brownout enabled approach in details, while the experimental results of the proposed approach are illustrated in Section 3.5. The summary are given in Section 3.6.

3.2 Related Work

It is an essential requirement for Cloud providers to reduce energy consumption, as it can both decrease operating costs and improve system reliability. Data centers can consume from 10 to 100 times more power per square foot than a typical office building. A large body of literature has focused on reducing energy consumption in cloud data centers, and the dominant categories for solving this problem are VM consolidation and Dynamic Voltage Frequency Scaling (DVFS) [80].

VM consolidation is regarded to be an act of combining into an integral whole, which helps to minimize the energy consumed by allocating work among fewer machines and turning off unused machines [123]. Under this approach, the VMs hosted on underutilized hosts would be consolidated to other servers and the remaining hosts would be transformed into the power-saving state. Beloglazov et al. [33] proposed scheduling algorithms considering Quality of Service and power consumption in Clouds. The algorithm's objective is energy-efficient mapping VMs to cloud servers through dynamic VM consolidation. The VM consolidation process is modeled as a bin-packing problem, where VMs are regarded as items and servers are regarded as bins. The advantages of the proposed algorithms are that they are independent of workload types and do not need to know the VM application information in advance.

The authors in [34] introduced adaptive approaches for VM consolidation with live migration according to VM historical data. Similar to [33], the VM placement is also modeled as a bin-packing problem, in which VMs from over-utilized servers are allocated to the PM with the least increase of power consumption and under-utilized servers are switched to be off or low power mode. In comparison to [33], this work considers multiple dimension resource (CPU, memory and bandwidth) and focuses more on VM placement optimization stage by proposing various policies. This work advances previous work by discussing online algorithm competitive ratio for energy efficient VM consolidation, which proves the algorithm's efficiency.

A self-adaptive method for VM consolidation on both CPU and memory is introduced in [105]. Its objective is minimizing the overall costs caused by energy-related issues. The VM assignment and migration processes are determined by probabilistic functions (Bernoulli trial). The mathematical analysis and realistic testbed results show that the proposed algorithm reduces total energy consumption for both CPU-intensive and memory-intensive workloads with suitable Bernoulli trial parameters. Compared with the bin-packing approach (adopted in [33] [34]), the proposed algorithm in this work can reduce migration times of VMs and its time complexity is lower than the bin-packing-based algorithm, which offers higher efficiency in the online scenario. To achieve the best performance, the disadvantage of this work is that it needs some efforts to find the most suitable Bernoulli trial parameter. Chen et al. [43] extended [105] and proposed another utilization-based probabilistic VM consolidation algorithm that aimed at reducing energy consumption and VM migration times. The author also made performance comparison with the algorithms in [33].

Corradi et al. [47] considered VM consolidation in a more practical viewpoint related to power, CPU and networking resource sharing and tested VM consolidation in OpenStack, which shows VM consolidation is a feasible solution to reduce energy consumption. Salehi et al. [129] proposed a VM consolidation based approach, which is an adaptive energy management policy that preempts VMs to reduce the energy consumption according to user-specific performance constraints and used fuzzy logic for obtaining appropriate decisions.

Li et al . [91] developed a Bayesian network-based estimation model for live VM migration, which enables the comprehensive configurations of nine parameters in real data centers. The objective of this approach is degrading energy consumption while avoiding inefficient VM migrations and improving QoS. The proposed approach takes advantage of the Bayesian network to model complex uncertainty systems and reduce the difficulty of knowledge access.

Dou et al. [56] proposed an energy-aware VM scheduling approach for QoS enhancement in cloud computing for big data applications. Focusing on big data applications, the authors aim to improve the QoS of big data services based on VM migrations while considering energy consumption. The motivation of this work is migrating the computational tasks to servers with lower energy consumption or servers with higher performance, which reduces service costs or execution time. The advantage of this work is that it considers two-level dynamic scheduling including tasks scheduling and VM scheduling for big data applications.

Han et al. [71] used Markov Decision Process (MDP) to handle VM management to reduce data center energy consumption. Through MDP, the optimal result is obtained by solving objective function. However, its solution dimension is quite large, the authors also proposed an approximate MDP approach to reduce the solution space and achieve faster convergence. In this approximate algorithm, a centralized controller calculates the utility function for each VM and determines the possibilities for the state transition. The state transitions in this algorithm represent the VMs are migrated from one server to another. The authors also theoretically validated the upper bound of the algorithm's error.

A practical OpenStack framework was implemented in [35] considering VM consolidation and data center power management. This framework is available for customized algorithm implementation. With public APIs, the framework is transparent to the original OpenStack installation, and it is not required to modify any OpenStack's configurations. This work is the first step to implement VM consolidation in OpenStack to minimize total energy consumption.

The DVFS technique introduces a trade-off between computing performance and energy consumed by the server. The DVFS technique lowers the frequency and voltage when the processor is lightly loaded, and utilizes maximum frequency and voltage when the processor is heavily loaded. Von et al. [146] introduced a power-aware scheduling algorithm based on DVFS-enabled cluster. Hanumaiah [72] et al. introduced a solution that considers DVFS, thread migration and active cooling to control the cores to maximize overall energy efficiency.

The authors in [83] modeled real-time service to be real-time VM requests and applied several DVFS algorithms to reduce energy consumption. Their objective is balancing the energy consumption and prices. The major concern in this work is that less energy is preferred at the same price, thus three different schemes based on DVFS are proposed to balance energy consumption and prices. The proposed schemes are easy to implement while the adaptive DVFS evaluations are restricted by the simplified and known-in-advance queuing model.

Deng et al. [51] proposed a method named CoScale for DVFS coordinating on CPU and memory while investigating performance constraints, which is the first trial to coordinate them together. Its objective is finding the most efficient frequency from a set of frequency settings while ensuring system performance. The most efficient frequencies for cores and memory are selected as they minimize the whole system energy consumption. CoScale adopts a fine-grained heuristic algorithm that iteratively predicates the component frequencies according to its performance counters as well as online models. However, CoScale is not suitable for offline workloads because it cannot reduce the possible frequency space as like in online workloads.

Teng et al. [140] combined DVFS and VM consolidation together to minimize total energy consumption. The energy saving objective is mainly applied to batch-oriented scenario, in which the authors introduced a DVFS-based algorithm to consolidate VMs on servers to minimize energy consumption and ensure job Service Level Agreement.

With theoretical analysis and realistic testbed on Hadoop, the authors proved that the proposed algorithm can find the most efficient frequency that is only associated with the processor type and its VM consolidation performance is insensitive to tunable parameters. The limitation of this work is that its realistic testbed is already upgraded to a new version that provides better management, which is more persuasive to implement the proposed approach on the updated platform.

Brownout was originally applied to prevent blackouts through voltage drops in case of emergency. Klein et al. [85] firstly borrowed the approach of brownout and applied it to cloud applications, aiming to design more robust applications under unpredictable loads. Tomas et al. [142] used the brownout along with overbooking to ensure graceful degradation during load spikes and avoid overload. Durango et al. [58] introduced novel load balancing strategies for applications by supporting brownout. In a brownout-compliant application or service, the optional parts are identified by developers and a control knob called **dimmer** that controls these optional parts is also exported. The dimmer value represents a certain probability given by a control variable and shows how often these optional parts are executed. In addition, a brownout controller is also required to adjust the dimmer value to avoid overload [142].

To the best of our knowledge, our approach is the first research to reduce energy consumption with brownout at components level, which also considers revenues for cloud service providers. Our approach provides a complementary option apart from VM consolidation and DVFS.

3.3 Problem Statement

In this section, we explain our system model and state the problem we aim to tackle. For reference, Table 3.1 summaries the symbols and their definitions throughout this chapter.

3.3.1 System Model

Our system model (Figure 3.1) includes several entities: users, applications and cloud providers, which are discussed below:

Users: Users submit service requests to cloud data centers to process. User entities contain user id and requested applications (services) information.

Applications: The application entities in our model come into system together with user entities. The applications consist of a number of components, which are tagged as mandatory or optional.

Mandatory component: The mandatory component is always running (activated) when the application is executed.

Optional component: The optional component can be set as activated or deactivated. These components have parameters like *utilization* and *discount* (penalty payment amount). Utilization indicates the amount of reduced utilization, and discount represents the price that is cut. The deactivation and activation of optional components are controlled by the **brownout controller**, which makes decisions based on system status and component selection policies.

The components can also be *connected*, which means that they communicate with each other and there are data dependencies between them. Therefore, we consider that if a component is deactivated, then all its connected optional components would also be set as deactivated. For example in Figure 3.1, if Com3 in Application #1 is deactivated, Com2 should also be deactivated; in Application #2, if Com1 is deactivated, Com3 should also be deactivated; in Application #n, if Com4 is deactivated, Com3 should also be deactivated, but Com2 is still working (Com1 is connected with Com3, but Com1 is mandatory, so it is not deactivated).

Cloud Providers: Cloud providers offer resources to meet service demands, which host a set of VMs or containers to run applications.

3.3.2 Power Model

To calculate the total energy consumption of data center, we adopt the server power model proposed by Zheng et al. [155]. The server power consumption is modeled as:

$$P_i^{server} = \begin{cases} P_i^{idle} + \sum_{j=1}^{w_i} u(VM_{i,j}) \times P_i^{dynamic} & , w_i > 0 \\ 0 & , w_i = 0 \end{cases} \tag{3.1}$$

Table 3.1: Symbols and definitions

Symbol	**Definition**
h_i	Server (host) i
P_i^{server}	Power of h_i
P_i^{idle}	Power when h_i is idle
$P_i^{dynamic}$	Power when h_i is fully loaded
P_i^{max}	Maximum power of h_i
hl	Server list in data center
w_i	Number of VMs assigned to h_i
$VM_{i,j}$	VM j on h_i
$u(VM_{i,j})$	Utilization of VM j on h_i
$d(VM_{i,j})$	Discount of VM j on h_i
App_c	Application component c
A_j	Total number of application components
$u(App_c)$	Utilization of application component c
$d(App_c)$	Discount of application component c
D_i	Total discount from server i
N	Total number of VMs
M	Total number of servers
Eff_{pa}	Algorithm efficiency of proposed algorithm pa
E_{pa}	Energy consumption of proposed algorithm pa
E_{bl}	Energy consumption of baseline algorithm bl
D_{pa}	Discount amount of proposed algorithm pa
α	Weight of discount to calculate algorithm efficiency
t	Time interval t
T	The whole scheduling interval
TP	Overloaded power threshold
θ_t	Dimmer value in brownout at time t
n_t	Number of overloaded hosts at time t
P_i^r	Expected power reduction of h_i
$COH()$	Calculate the number of overloaded hosts
$HPM()$	Host power model to compute expected utilization reduction
$u_{h_i}^r$	Expected utilization reduction on hi
$VUM()$	VM utilization model to compute expected utilization reduction
$u_{VM_{i,j}}^r$	Expected utilization reduction on VM_j on h_i
$CSP()$	Component selection policy to deactivate components
$dcl_{i,j,t}$	Deactivated component list at time t on h_i
S_t	Set of deactivated components connection tags
$Ct(App_c)$	Connection tag of component App_c
$ocl_{i,j,t}$	Optional component list of VM_j on h_i at time t
p	Position index in optional component list
R_h	Available resource of host
R_v	Maximum requested resource of VM
C_e	Cost of energy consumption per unit of time
C_o	Cost of overloads per unit of time
ε	Relative cost of overloads compared with C_e
t_b	Time for brownout operation
t_m	Time for VM consolidation
τ	The times of brownout and VM consolidation occur in T

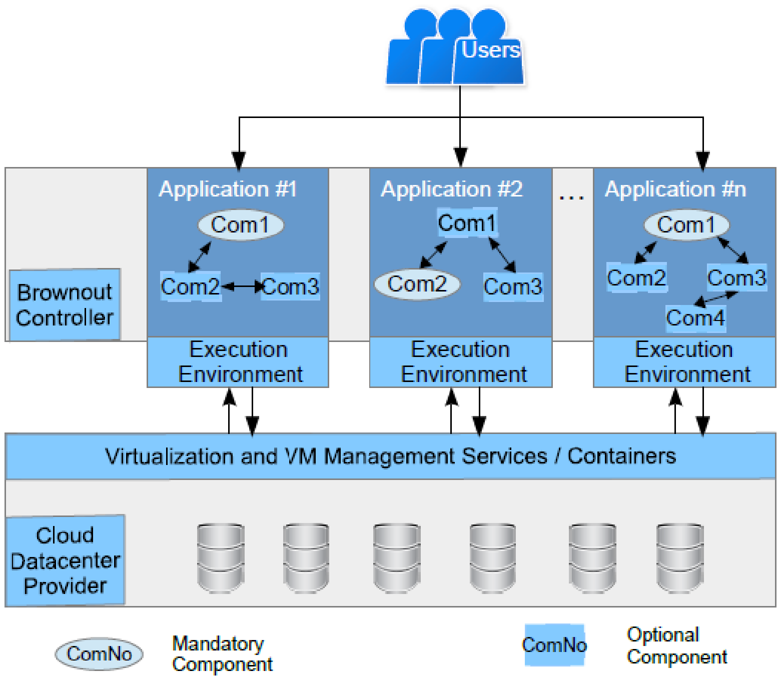


Figure 3.1: System model with brownout

P_i^{server} is composed of idle power and dynamic power. The idle power is regarded as constant and the dynamic power is linear to the total CPU utilization of all the VMs on the server. If no VM is hosted on a server, the server is turned off to save power. $VM_{i,j}$ refers to the jth VM on the ith server, w_i means the number of VMs assigned to server i.

$$u(VM_{i,j}) = \sum_{c=1}^{A_j} u(App_c) \tag{3.2}$$

The utilization of $VM_{i,j}$ is represented as $u(VM_{i,j})$, which is computed by summing up all the application utilization on the jth VM. The c is the component id and A_j is the number of application components. As processors are still the main energy consumer of servers, we focus on CPU utilization in this work.

Note: We consider the application component can be activated/deactivated within a rather short time compared with the scheduling period. Thus, in our model, we assume that the time and energy cost for the activation/deactivation operations are 0.

3.3.3 Discount Amount

$$D_i = \sum_{j=1}^{w_i} d(VM_{i,j}) \tag{3.3}$$

In Equation 3.3, D_i is the total discount amount obtained from all VMs, in which the individual discount $d(VM_{i,j})$ from $VM_{i,j}$ is the sum of all application components discount amount $d(App_c)$ as shown in Equation 3.4.

$$d(VM_{i,j}) = \sum_{c=1}^{A_j} d(App_c) \tag{3.4}$$

A_j is the number of applications hosted on VM_j, and $d(VM_{i,j})$ is the discount happened from VM_j on server i, and D_i is the total discount amount on server i.

3.3.4 Constraints and Objectives

The above equations subject to the following constraints:

$$\sum_{i=1}^{M} w_i = N \tag{3.5}$$

$$\sum_{j=1}^{w_i} u(VM_{i,j}) \leq 1, \forall i \in [1, M] \tag{3.6}$$

N is the total number of VMs and M is the total number of servers. Equation 3.5 represents the total number of VMs assigned to hosts w_i equals to the sum of VMs. Equation 3.6 represents the sum of all the VMs utilization cannot surpass their host available utilization.

We formulate the objectives of this problem as:

$$\min \sum_{i=1}^{M} P_i^{server} \tag{3.7}$$

As well as

$$\min(\sum_{i=1}^{M} D_i) \tag{3.8}$$

Therefore, we aim at investigating the trade-off total energy consumption and discount amount.

To measure the performance of an algorithm, we represent the algorithm efficiency

Eff_{pa}:

$$Eff_{pa} = \frac{E_{pa}}{E_b} + \alpha D_{pa} \tag{3.9}$$

where E_{pa} is the energy consumption of the proposed algorithm, E_b is the baseline algorithm energy consumption, D_{pa} is the discount amount offered by the proposed algorithm. If the proposed algorithm saves more energy than the baseline algorithm, $\frac{E_{pa}}{E_b}$ is a value between 0 to 1, and D_{pa} represents the offered discount percentage, which also belongs to 0 to 1. Thus, the smaller Eff_{pa} is, the more energy is reduced and less discount amount is offered. The α is the weight of discount, its default value is 1.0, if service provider care more on discount, α is set as larger than 1.0; if they care more about energy saving, α is set as less than 1.0.

3.4 Proposed Approach

Prior to brownout approach, we require a VM placement and consolidation algorithms. We adopt the placement and consolidation algorithm (PCO) proposed by Beloglazov et al. [33]. Then we propose our brownout enabled algorithm based on PCO and introduce a number of component selection policies considering component utilization and discount.

3.4.1 VM Placement and Consolidation Algorithm (PCO)

The VM placement and consolidation (PCO) algorithm is an adaptive heuristics for dynamic consolidation of VMs and extensive experiments show that it can significantly reduce energy consumption. In the initial VM placement phase, PCO sorts all the VMs in decreasing order of their current CPU utilization and allocates each VM to the host that increases the least power consumption due to this allocation. In the VM consolidation phase, PCO optimizes VM placement according to loads of hosts: PCO separately picks VMs from over-utilized and under-utilized hosts to migrate, and finds new placements for them. After migration, the over-utilized hosts are not overloaded any more and the under-utilized hosts are switched to sleep mode.

Algorithm 3.1: Energy Efficient with Brownout Algorithm (EEBA)

```
Input : hostList $hl$ with size $M$, application components information, overloaded power threshold
        $TP$, dimmer value $\theta_t$ at time $t$, scheduling interval $T$, deactivated component list $dcl_{i,j,t}$ of
        $VM_{i,j}$ on host $h_i$, power model of host $HPM$, VM utilization model $VUM$, component
        selection policy $CSP$
Output: total energy consumption, discount amount, number of shutting down hosts
use PCO algorithm to initialize VMs placement
initialize parameters with inputs, like $TP$
for $t \leftarrow 0$ to $T$ do
    $n_t \leftarrow COH(hl)$
    if $n_t > 0$ then
        $\theta_t \leftarrow \sqrt{\frac{n_t}{M}}$
        forall $h_i$ in $hl$ (i.e. $i = 1, 2, \ldots, M$) do
            if ($P_i^{server} > P_i^{max} \times TP$) then
                $P_i^r \leftarrow \theta_t \times P_{h_i}$
                $u_{h_i}^r \leftarrow HPM(h_i, P_i^r)$
                forall $VM_{i,j}$ on $h_i$ (i.e. $j = 1, 2, \ldots, w_i$) do
                    $dcl_{i,j,t} \leftarrow$ NULL
                    $u_{VM_{i,j}}^r \leftarrow VUM(u_{h_i}^r, VM_{i,j})$
                    $dcl_{i,j,t} \leftarrow CSP(u_{VM_{i,j}}^r)$
                    $D_i \leftarrow D_i + d(VM_{i,j})$
                end
            end
        end
    else
        activate deactivated components
    end
    use VM consolidation in PCO algorithm to optimize VM placement
end
```

3.4.2 Energy Efficient Brownout Enabled Algorithm

Our proposed energy efficient brownout enabled approach (noted as **EEBA**) is an enhanced approach based on PCO algorithm. According to host power consumption, the brownout controller dynamically deactivates or activates applications' optional components on VMs to relieve overloads and reduce the power consumption.

As shown in Algorithm 3.1, EEBA mainly consists of 6 steps:

Before entering the approach procedures, service provider firstly needs to initialize VM placement by algorithm like PCO and overloaded power threshold (lines 1-2). The power threshold TP is a value for checking whether a host is overloaded. Then the other steps are as below:

1) In each time interval t, checking all the hosts and counting the number of overloaded hosts as n_t (line 4);

2) Adjusting dimmer value θ_t as $\sqrt{\frac{n_t}{M}}$ based on the number of overloaded hosts n_t and host size M (line 6).

As mentioned in our related work, the dimmer value θ_t is a control knob used to determine the adjustment degree of power consumption at time t. The dimmer value θ_t is 1.0 if all the hosts are overloaded at time t and it means that brownout controls components on all the hosts. The dimmer value is 0.0 if no host is overloaded and brownout will not be triggered at time t. The dimmer adjustment approach shows that dimmer value varies along with the number of overloaded hosts.

3) Calculating the expected utilization reduction on the overloaded hosts (lines 8-10). According to the dimmer value and host power model, EEBA calculates expected host power reduction P_i^r (line 9) and expected utilization reduction $u_{h_i}^r$ (line 10). With host power model (like in Table 3.3), we have host power at different utilization levels, so the utilization reduction can be computed based on power reduction. For example, in a power model, the host with 100% utilization is 247 Watts and 80% utilization is 211 Watts, if the power is required to be reduced from 247 to 211 Watts, the expected utilization reduction is $100\% - 80\% = 20\%$.

4) Calculating the expected utilization reduction on VM (lines 11-13). An empty deactivated component list $dcl_{i,j,t}$ of VM_j on host h_i is initialized to prepare for storing deactivated components (line 12). Then the expected VM utilization reduction $u_{VM_{i,j}}^r$ is computed based on VM utilization model as VM utilization multiplies $u_{h_i}^r$ (line 13).

5) Applying component selection policy CSP to find and deactivate components list $dcl_{i,j,t}$ (line 14). According to the expected VM utilization reduction $u_{VM_{i,j}}^r$, component selection policy is responsible for finding the components satisfying the utilization constraint, deactivating these components and their connected ones, and updating total discount amount (line 15).

6) In EEBA, if no host is above the power threshold, the algorithm activates the optional components that have been set as deactivated (line 20).

Finally, after finishing the main steps of EEBA, VM consolidation in PCO algorithm is applied to optimize VM placement (line 22).

The EEBA algorithm takes effect between the VM placement and VM consolidation in PCO. VMs are initially placed by VM placement phase in PCO, after that, if no host is above the power threshold, the EEBA does not work; otherwise, the brownout is triggered to handle the overloaded condition, then the VM consolidation phase in PCO is applied.

As applications may have multiple optional components with different utilization and discount amount, for Algorithm 3.1 step 4 that applies component selection policy, we have designed several policies:

Nearest Utilization First Component Selection Policy (NUFCS): The objective of NUFCS is finding and deactivating a single component in the component list. Compared with other components, the single component has the nearest utilization to $u^r_{VM_{i,j}}$. NUFCS can find the goal component in $O(n)$ time, which is efficient in online scheduling.

If the deactivated component is connected with other components, NUFCS also deactivates other connected components. NUFCS runs fast and can reduce utilization, but if $u^r_{VM_{i,j}}$ is much larger than all the single component utilization in the component list, more components should be deactivated to achieve expected energy reduction. Therefore, we propose another 3 multiple components selection policies to achieve expected utilization reduction:

Lowest Utilization First Component Selection Policy (LUFCS): LUFCS selects a set of components from the component with the lowest utilization until these components achieve expected utilization reduction. This policy follows the assumption that the component with less utilization is less important for users. Hence, with this policy, the service provider deactivates a number of components with low utilization to satisfy the expected utilization reduction.

Lowest Price First Component Selection Policy (LPFCS): LPFCS selects a set of components from the component with the lowest discount. This policy focuses more on discount and its objective is deactivating a number of components with less discount amount and satisfying the expected utilization reduction.

Highest Utilization and Price Ratio First Component Selection Policy (HUPRFCS): HUPRFCS selects a set of components considering component utilization and discount together. The components with larger $\frac{u(App_c)}{d(App_c)}$ values are prior to be selected. Its objective is deactivating the components with higher utilization and smaller discount. Therefore, the service provider saves more energy while offering less discount amount.

Algorithm 3.2: Component Selection Policy: Lowest Utilization First Component Selection Policy (LUFCS)

```
Input  : expected utilization reduction $u^r_{VM_{i,j}}$ on $VM_{i,j}$
Output: deactivated components list $dcl_{i,j,t}$
1  Sort the optional component list $ocl_{i,j,t}$ based on utilization $u(App_c)$ in ascending order //Other
     policies may change the sorting approach at this line. If there are connected components, the
     connected components are treated together and sorted by their average utilization
2  $S_t \leftarrow$ NULL
3  if $u(App_{1st}) \geq u^r_{VM_{i,j}}$ then
4      $dcl_{i,j,t} \leftarrow dcl_{i,j,t} + App_{1st}$
5      $S_t \leftarrow S_t + Ct(App_{1st})$
6      forall $App_c$ in $ocl_{i,j,t}$ do
7          if $Ct(App_c)$ is in $S_t$ then
8              $dcl_{i,j,t} \leftarrow dcl_{i,j,t} + App_c$
9              $d(VM_{i,j}) \leftarrow d(VM_{i,j}) + d(App_c)$
10         end
11     end
12 else
13     $p \leftarrow 0$
14     forall $App_c$ in $ocl_{i,j,t}$ do
15         if ($\sum_0^k(App_c) < u^r_{VM_{i,j}}$ & $\sum_0^{k+1}(App_c) > u^r_{VM_{i,j}}$) then
16             if ($u^r_{VM_{i,j}} - \sum_0^k(App_c) < \sum_0^{k+1}(App_c) - u^r_{VM_{i,j}}$) then
17                 $p = k - 1$
18             else
19                 $p = k$
20             end
21         end
22         break
23     end
24     for $c \leftarrow 0$ to $p$ do
25         $dcl_{i,j,t} \leftarrow dcl_{i,j,t} + App_c$
26         $S_t \leftarrow S_t + Ct(App_c)$
27         $d(VM_{i,j}) \leftarrow d(VM_{i,j}) + d(App_c)$
28     end
29     forall $App_c$ in $ocl_{i,j,t}$ do
30         if $Ct(App_c)$ in $S_t$ then
31             $dcl_{i,j,t} \leftarrow dcl_{i,j,t} + App_c$
32             $d(VM_{i,j}) \leftarrow d(VM_{i,j}) + d(App_c)$
33         end
34     end
35 end
36 return $dcl(i, j, t)$
```

Algorithm 3.2 shows an example about how the component selection policy works. The example is about LUFCS: the input of the algorithm is the expected configured utilization $u^r_{VM_{i,j}}$ and the output is the deactivated components list $dcl_{i,j,t}$. The steps are:

a) Algorithm 3.2 sorts the optional components list $ocl_{i,j,t}$ based on component utilization parameter in ascending sequence (line 1), therefore, the component with the lowest utilization is put at the head. For connected components, the sorting process is modified as treating the connected components together and using their average utilization for sorting, which lowers the priority of deactivating connected components to avoid deactivating too many components due to connections;

b) Initialize a set S_t that stores the deactivated components connection tags (line 2);

c) Algorithm 3.2 deactivates the first component and its connected components if it satisfies the expected utilization reduction (lines 3-11). If the first component utilization parameter value is above $u^r_{VM_{i,j}}$, Algorithm 3.2 puts this component into the deactivated components list $dcl_{i,j,t}$ and puts its connection tag $Ct(App_1st)$ (a tag shows how it is connected with other components) into S_t. After that, Algorithm 3.2 finds other connected components and put them into deactivated components list. Finally, summing up the deactivated components discount amount as $d(VM_{i,j})$;

d) If the first component utilization does not satisfy the expected utilization reduction, Algorithm 3.2 finds a position p in the optional components list (lines 13-23). The sublist before $p - 1$ is the last sublist that makes its components utilization sum less than $u^r_{VM_{i,j}}$ and the sublist that before p is the first sublist that makes its components utilization sum larger than u. The policy selects the sublist with utilization sum closer to the $u^r_{VM_{i,j}}$ from these two sublists;

e) Algorithm 3.2 puts all the components in the sublist into the deactivated components list and puts their connection parameters into the S_t (lines 24-28);

f) Algorithm 3.2 finds other connected components and puts them into the deactivated components list, and updates the discount amount (lines 29-35);

g) Finally, Algorithm 3.2 returns the deactivated components list (line 36).

The LPFCS and HUPRFCS procedures are quite similar to LUFCS except the sorting process at line 1. For example, the LPFCS sorts the optional components list according to component discount, while HUPRFCS sorts the optional components list based on component utilization and discount ratio $\frac{u(App_c)}{d(App_c)}$. For connected components, these policies also treat them together and use their discount or utilization and discount ratio to sort.

The complexity of our proposed algorithm at each time interval is calculated based on two parts, one is the brownout part and the other is the PCO part. At each time interval, the complexity of the brownout part is $O(m * M)$, where m is the maximum number of components in all applications, M is the number of hosts. The complexity of the PCO part is $O(2M)$ as analyzed in [33]. The complexity at each time interval of our proposed algorithm is the sum of the two parts, which is to $O((2 + m) * M)$.

3.4.3 EEBA Competitive Analysis

We apply competitive analysis [37][34] to analyze the brownout approach combining with VM consolidation for multiple hosts and VMs. We assume that there are M homogeneous hosts and N homogeneous VMs. If the available resource of each host is R_h and maximum resource that can be allocated to VM is R_v, then the maximum number of VMs allocated to host is $\frac{R_h}{R_v}$. Overloaded situation occurs when VMs require more capacity than R_h. The brownout approach handles with the overloaded situation with a processing time t_b, and VMs are migrated between hosts through VM consolidation with migration time t_m. The cost of overloads per unit of time is C_o, and the cost of energy consumption is C_e. Without loss of generality, we can define $C_e = 1$ and $C_o = \varepsilon$. Then we have the following theorem:

Theorem 1. *The upper bound of the competitive ratio of EEBA algorithm for the components control and VM migration problem is* $\frac{EEBA(I)}{OPT(I)} \leq 1 + \frac{N\varepsilon}{N+M}$.

Proof: The EEBA controls the application components to handle with the overloaded situation and applies VM consolidation to reduce energy consumption. This algorithm deactivates application components to make the hosts to be not overloaded and consolidates VMs to the minimum number of hosts. Under normal status, the number of VMs allocated to each host is $\frac{N}{M}$, while in overloaded situation, at least $\frac{N}{M} + 1$ VMs are allocated to a single host. Thus, the maximum number of overloaded hosts is $M_o = \lfloor \frac{N}{\frac{N}{M}+1} \rfloor$, which is equivalent to $M_o = \lfloor \frac{MN}{N+M} \rfloor$.

In the whole scheduling interval T, we split the time into 3 parts $T = (t_b + t_m)\tau +$

t_0, where t_b is the time that EEBA uses brownout to relieve overloads, t_m is the time consumed for VM migration, t_0 is the time that hosts running at normal status and $\tau \in \mathbb{R}^+$. For the brownout and VM migration parts, the behaviors are as below:

1). During the t_b, the brownout controller selects application components on overloaded hosts and deactivates them. Because all the hosts are active during t_b, the cost of this part is $t_b(MC_e + M_oC_o)$.

2). During the t_m, if there are still overloaded hosts, VMs are migrated from the overloaded hosts $M_o^{'}$, and $M_o^{'} \leq M_o$. As the VM migration time is t_m and all the hosts are active during migration, the total cost during this time of period is $t_m(MC_e + M_o^{'}C_o)$.

Therefore, the total cost C during $t_b + t_m$ is defined as below:

$$C = t_b(MC_e + M_oC_o) + t_m(MC_e + M_o^{'}C_o). \tag{3.10}$$

And the total cost incurred by EEBA for the input I is shown in Equation 3.11:

$$EEBA(I) = \tau[t_b(MC_e + M_oC_o) + t_m(MC_e + M_o^{'}C_o)] \tag{3.11}$$

The optimal offline algorithm for this problem only keeps the VMs at each host and does not apply brownout and VM consolidation. Therefore, the total cost of an optimal offline algorithm is defined as:

$$OPT(I) = \tau(t_b + t_m)MC_e \tag{3.12}$$

Then we compute the competitive ratio of an optimal offline deterministic algorithm as:

$$\frac{EEBA(I)}{OPT(I)} = \frac{\tau[t_b(MC_e + M_oC_o) + t_m(MC_e + M_o^{'}C_o)]}{\tau(t_b + t_m)MC_e} \tag{3.13}$$

As $M_o^{'} \leq M_o$, we have:

$$\frac{EEBA(I)}{OPT(I)} \leq \frac{\tau[(t_b + t_m)(MC_e + M_oC_o)]}{\tau(t_b + t_m)MC_e} = \frac{MC_e + M_oC_o}{MC_e} \tag{3.14}$$

As $M_o = \lfloor \frac{MN}{N+M} \rfloor$, we have $M_0 \leq \frac{MN}{N+M}$ and combine with Equation 3.14 as well $C_e =$

Table 3.2: Host / VM Types and capacity

Name	CPU	Cores	Memory	Bandwidth	Storage
Host Type 1	1.86 GHz	2	4 GB	1 Gbit/s	100 GB
Host Type 2	2.66 GHz	2	4 GB	1 Gbit/s	100 GB
VM Type 1	2.5 GHz	1	870 MB	100 Mbit/s	1 GB
VM Type 2	2.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 3	1.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 4	0.5 GHz	1	613 MB	100 Mbit/s	1 GB

Table 3.3: Power consumption of servers in Watts

Servers	**0% (sleep mode)**	**10%**	**20%**	**30%**	**40%**	**50% (idle)**	**60%**	**70%**	**80%**	**90%**	**100% (max)**
IBM x3550 M3 (Intel Xeon X5670 CPU)	66	107	120	131	143	156	173	191	211	229	247
IBM x3550 M3 (Intel Xeon X5675 CPU)	58.4	98	109	118	128	140	153	170	189	205	222

$1, C_o = \varepsilon$, the competitive ratio is defined as:

$$\frac{EEBA(I)}{OPT(I)} \le \frac{MC_e + M_oC_o}{MC_e} \le \frac{M + \frac{MN}{N+M}\varepsilon}{M} = 1 + \frac{N\varepsilon}{N+M} \tag{3.15}$$

■

3.5 Performance Evaluation

3.5.1 Environment Setting

We use the CloudSim framework [40] to simulate a cloud data center with 100 hosts. Two types of hosts and four types of VMs are modeled based on current offerings in EC2 as shown in Table 3.2. The power models of hosts we adopted are derived from IBM System x3550 M3 with CPU Intel Xeon X5670 and X5675 [1], and their power consumption at different utilization levels are demonstrated in Table 3.3. We assume that the idle host consumes 50% of the full utilization, as mentioned in [106] that idle hosts can consume

Table 3.4: Parameter configurations for testing

Parameters	**P1: Optional component utilization threshold**	**P2: Percentage of optional Components**	**P3: Percentage of connected components**	**P4: Discount**
Range	0% to 100%	0% to 100%	0% to 100%	0% to 100%
Categories	25%, 50%, 75%, 100%	25%, 50%, 75%, 100%	25%, 50%, 75%, 100%	varying with P1

Table 3.5: A testcase example

Testcase ID	Optional component utilization threshold	Percentage of optional Components	Percentage of connected components	Discount
TC1	50%	50%	25%	50%

up to 60% of peak utilization.

The application modeled in CloudSim is based on a class called cloudlet. We have extended the cloudlet to model application with optional components, and each component has its corresponding CPU utilization, discount amount, and connection parameter. The components are uniformly distributed on VMs.

We adopt the realistic workload trace from more than 1000 PlanetLab VMs [122] to create an overloaded environment [32]. Our experiments are simulated under one-day scheduling period and repeated 10 times based on 10 different days PlanetLab data. The brownout is invoked every 300 seconds (5 minutes per time slot) if hosts power surpasses the power threshold. The CPU resource is measured with the capacity of running instructions. Assuming that the application workload occupies 85% resource on a VM and the VM has 1000 million instructions per second (MIPS) computation capacity, then it presents the application constantly requires $0.85 \times 1000 = 850$ MI to $1.0 \times 1000 = 1000$ MI per second in the 5 minutes.

To reflect the impact of different configurations, we investigate a set of parameters as shown in Table 3.4:

1) Optional component utilization threshold: it represents the threshold portion of utilization that is optional and can be reduced by deactivating optional components. An optional component with 25% utilization means 25% of application utilization is reduced if it is set as deactivated. We adjust this parameter from 0% to 100% and categorize it as 25%, 50%, 75% and 100%.

2) Percentage of optional components: it represents how many components of the total components are optional. Assuming the number of all components is num_{com} and the number of optional components is num_{opt}, then the percentage of optional components is $\frac{num_{opt}}{num_{com}}$. This parameter is varied from 0% to 100% and is categorized as 20%, 50%, 75% and 100%.

3) Percentage of connected components: it represents how many components are con-

nected among all the components. Assuming the number of connected components is $num_{connected}$, then the percentage of connected components is $\frac{num_{connected}}{num_{com}}$. This parameter is also varied from 0% to 100% and is categorized as 25%, 50%, 75% and 100%. The connections between components are randomly generated based on the percentage of connected components.

4) Discount: It represents the discount amount that allowed to be paid back to the user if components are deactivated. We assume that application maximum discount is identical to the optional component utilization threshold, for example, 50% optional component utilization threshold comes along with 50% discount.

We assume that the components utilization $u(App_c)$ and discount $d(App_c)$ conform normal distribution $u(App_c) \sim N(\mu, \sigma^2)$, $d(App_c) \sim N(\mu, \sigma^2)$, the μ is the mean utilization of component utilization or discount, which is computed as the optional component utilization threshold (or discount amount) divided by the number of optional components. The σ^2 is the standard deviation of components utilization or discount.

Based on σ^2, we consider two component design patterns according to component utilization and discount. One pattern is that components are designed with uniform or approximate utilization and discount, which means each component is designed to require same or approximate resource amount, like there are 5 components and each component requires 10% utilization and offers 10% discount. We define the components as **approximate** if their utilization standard deviation and discount standard deviation are both less than 0.1. Another pattern is that components utilization and discount are conspicuous different, which means the components are designed to require quite different resource. We define the components as **different** if either their utilization standard deviation or discount standard deviation is larger than 0.1.

According to Table 3.4, Table 3.5 shows a testcase with configured parameters, the optional component utilization threshold is configured as 50%, the percentage of optional utilization is configured as 50%, the percentage of connected components is set as 25% and the discount is 50%.

Table 3.6 demonstrates an application components example fits the configurations in Table 3.5. This application consists of 8 components: 4 of them (50%) are optional components. Each component has utilization, discount and connected relationship with other components: the optional component utilization threshold is 50% (the utilization sum of component 5, 6, 7 and 8), there are 2 components (20%) of all components are connected (component 5 and 6) and the total discount of optional components is 50%.

3.5.2 Results and Analysis

In this section, we compare EEBA performance with two baselines algorithms:

1) **VM Placement and Consolidation algorithm (PCO)**: the algorithm is described in Section 3. 4.1. We configure its upper threshold as 0.8 and the lower threshold as 0.2.

Table 3.6: An application component example1

Components ID	Mandatory / Optional	Utilization	Discount	Connected
Comp 1	Mandatory	10%	10%	N/A
Comp 2	Mandatory	10%	10%	N/A
Comp 3	Mandatory	20%	20%	N/A
Comp 4	Mandatory	10%	10%	N/A
Comp 5	Optional	5%	5%	Comp8
Comp 6	Optional	10%	10%	Comp7
Comp 7	Optional	15%	20%	N/A
Comp 8	Optional	20%	15%	N/A

2) **Utilization-based Probabilistic VM consolidation algorithm (UBP)** [43]: in the VM placement, UBP adopts the same approach as PCO: sorting all the VMs in decreasing order based on their utilization and allocating each VM to the host that increases the least power consumption. In the VM consolidation phase, UBP applies a probabilistic method [105] to select VMs from the overloaded host. The probabilistic method calculates the migration probability based on PM utilization u as :

$$f_m(u) = (1 - \frac{u-1}{1-T_h})^\lambda \tag{3.16}$$

where $f_m(u)$ is the migration probability, T_h is the upper threshold for detecting overloads and λ is a constant to adjust probability. We configure the $T_h = 0.8$ and $\lambda = 1$.

In EEBA, we also configure $TP = 0.8$ that is as same as the upper threshold in PCO and T_h in UBP.

We separately conduct experiments for the two design patterns to evaluate algorithm performance. With approximate components, our proposed policies LUFCS, LPFCS and

HUPRFCS select the same components, so we focus on comparing PCO, UBP, NUFCS and LUFCS policies, which represent baseline algorithms without brownout, EEBA with single component selection policy and EEBA with multiple components selection policy respectively. While with different components, we focus on comparing the LUFCS, LPFCS and HUPRFCS policies to evaluate the performance of different multiple components selection policies.

In addition, as introduced in Section 3.3.1, components may be connected. Therefore, to investigate the effects of individual component selection and connected components selection, we separately run experiments for components without connections and connected components.

As PCO and UBP performance are not influenced by the parameters in Table 3.4, we firstly obtain their results as baselines. The PCO leads to 345.3 kWh with 95% confidence interval (CI): (336.9, 353.7), and UBP reduces this value to 328.5kWh with 95% CI: (321.1, 335.9). Both these two algorithms offer no discount and no disabled utilization. Because UBP performance is better than PCO, we set PCO as the benchmark. Referring to Equation 3.9, E_b is set as 345.3, so PCO algorithm efficiency $Eff_{PCO} = 345.3/345.3 + 0.0 = 1.0$; for UBP algorithm, its efficiency is $Eff_{UBP} = 321.1/345.3 + 0.0 = 0.95$.

Components without Connections

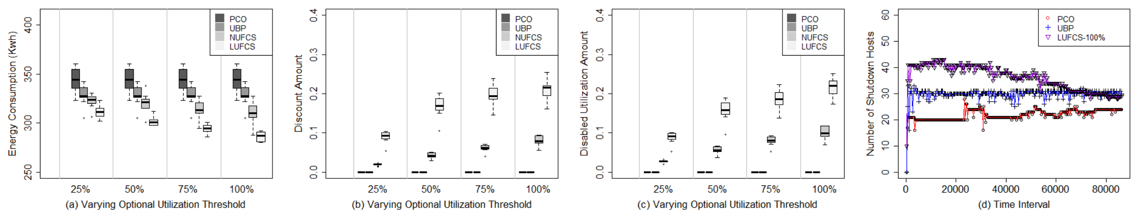


Figure 3.2: Comparison by varying optional utilization threshold for approximate components

1) Varying Optional Component Utilization Threshold

Figure 3.2 shows the comparison between PCO, UBP, NUFCS and LUFCS when components are approximate by varying the optional utilization threshold (the percentage of optional components is fixed as 50%). Figure 3.2a shows the energy consumption of these policies respectively. NUFCS and LUFCS can save more energy when the optional

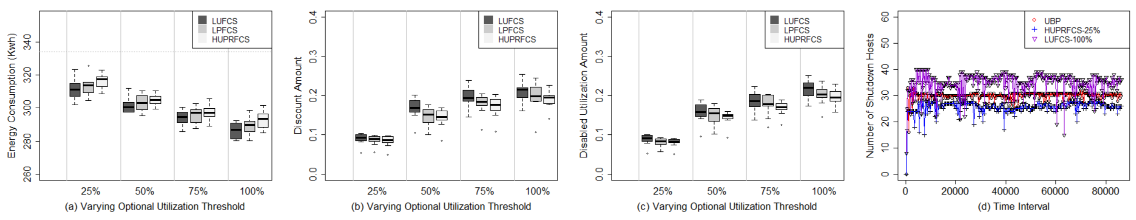


Figure 3.3: comparison by varying optional utilization threshold for different components

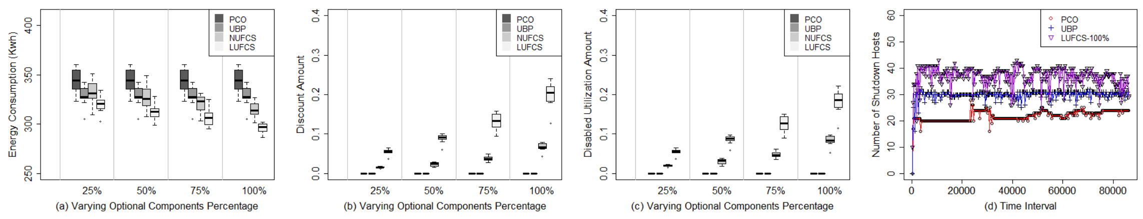


Figure 3.4: Comparison by varying optional component percentage for approximate components

component utilization threshold is larger. However, more discount amount is also offered to users according to Figure 3.2b. The reason lies in that Figure 3.2c and Figure 3.2d demonstrate that more utilization amount is disabled in NUFCS and LUFCS, and more hosts are shutdown by these policies, which contributes to more energy reduction. We use UBP and NUFCS with 100% optional utilization threshold to compare the number of shutdown hosts with PCO, which shows the maximum and minimum number of shutdown hosts in this series of experiments. The number of shutdown hosts of other experiments falls between the UBP and LUFCS-100% lines in Figure 3.2d. Compared with UBP, NUFCS reduces 2% to 7% energy and LUFCS reduces 6% to 21% energy while 1% to 6% and 8% to 22% discount amount are offered respectively.

Figure 3.3 shows LUFCS, LPFCS and HUPRFCS policies' effects on energy and discount amount when components are different and optional utilization threshold increases, more energy is reduced and more discount amount is offered. As Figure 3.3a and Figure 3.3a illustrate, when components are different, LUFCS cannot save as much energy as when components are approximate. For example, LUFCS-100% in Figure 3.3a shows it reduces maximum 15% energy (the dotted line represents UBP energy consumption), while LUFCS-100% in Figure 3.3a saves 21% energy. Therefore, our proposed policies work better when components are designed with approximate resource requirement,

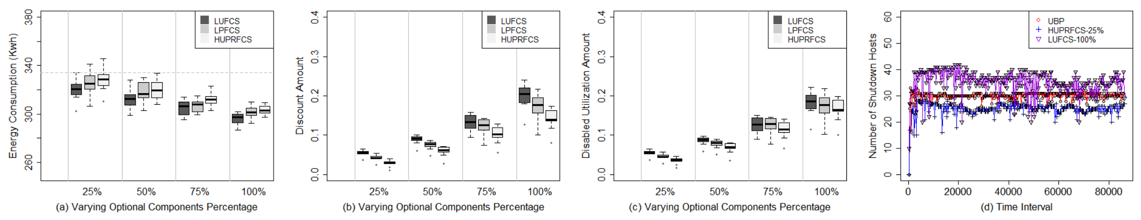


Figure 3.5: Comparison by varying optional component percentage for different components

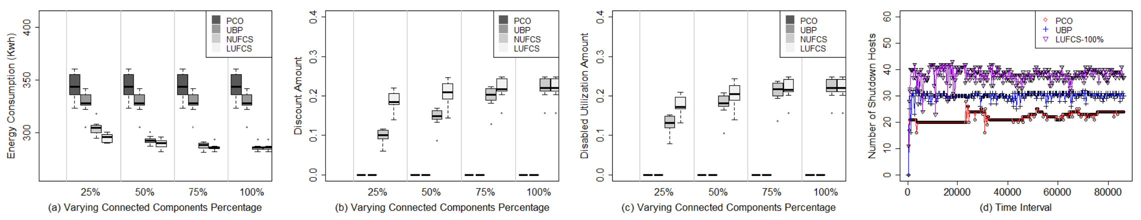


Figure 3.6: Comparison by varying percentage of connected components for approximate components

which also shows the value of proper design of components or microservices. According to Figure 3.3a and Figure 3.3b, LUFCS reduces the maximum energy, but also offers the maximum discount amount, which gives LUFCS $<$ LPFCS $<$ HUPRFCS in energy and LUFCS $>$ LPFCS $>$ HUPRFCS in discount amount. We conduct paired t-tests for energy consumption of these policies, when optional utilization threshold is 25%, the *p-value* for LUFCS-25% and HUPRFCS-25% is 0.12, which shows nonstatistically significant differences between these policies, while the optional utilization threshold increases, the *p-value* for LUFCS-50% and LPFCS-50% is 0.038 and the *p-value* for LPFCS-50% and HUPRFCS-50% is 0.046, which shows there are statistically significant differences in energy consumption. As shown in Figure 3.3c and Figure 3.3d, it reflects more utilization amount is disabled and more hosts are shutdown in LUFCS as shown. The different effects between these policies come from the LUFCS selects components without considering discount, as it can select as many components as possible until achieving expected utilization reduction. While other two policies consider discount amount and do not deactivate as many components as in LUFCS.

2) Varying Percentage of Optional Components

Figure 3.4 shows the results when components are approximate by varying the percentage of optional components (the optional component utilization threshold is fixed as

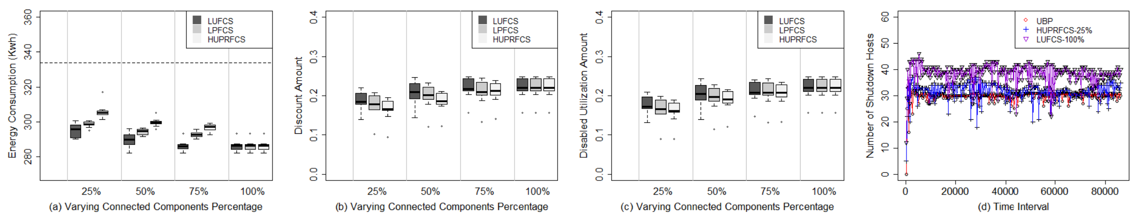


Figure 3.7: Comparison by varying percentage of connected components for different components

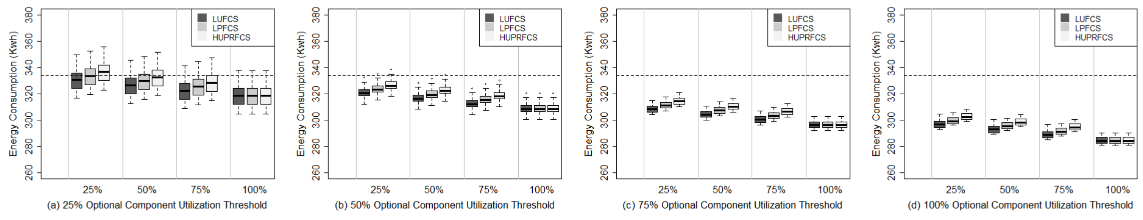


Figure 3.8: Energy consumption comparison by varying percentage of connected components and optional component utilization threshold

50%). Figure 3.4a and Figure 3.4b illustrate that in comparison to PCO and UBP, more energy is saved and more discount amount is offered with the increase of the optional components in NUFCS and LUFCS, which results from more options of components are available to be selected. In comparison to UBP, when more than 25% components are optional, NUFCS saves 1% to 7% energy and offers maximum 12% discount amount, and LUFCS saves 5% to 19% energy but offers 5% to 20% discount amount. As shown in Figure 3.4c and Figure 3.4d, compared with UBP, LUFCS disables maximum 19% utilization amount and more than 8 hosts averagely.

Figure 3.5 compares LUFCS, LPFCS and HUPRFCS policies for different components when varying the percentage of optional components. The results in Figure 3.5a and Figure 3.5b show that these policies save more energy when optional components increases, and show LUFCS $<$ LPFCS $<$ HUPRFCS in energy as well as LUFCS $>$ LPFCS $>$ HUPRFCS in discount amount. As demonstrated in Figure 3.5c and Figure 3.5d, LUFCS disables more utilization amount than other two policies and shuts down the maximum number of hosts when with 100% optional components. Through paired t-tests for 25% optional components, we observe the p-value for LUFCS and LPFCS is 0.035, which shows statistically significant differences. But the p-value for LPFCS and HUPRFCS is 0.095, which shows nonstatistically significant different. Similar p-values are also ob-

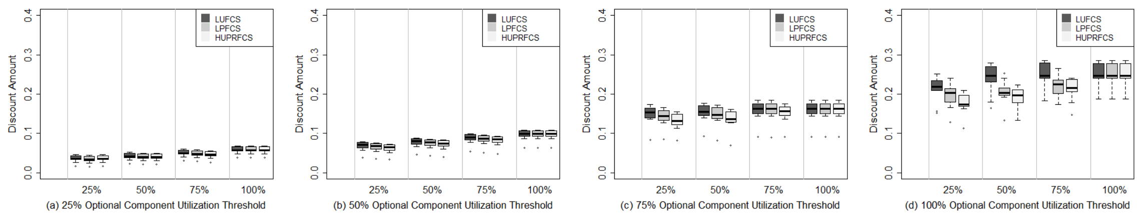


Figure 3.9: Discount amount comparison by varying percentage of connected components and optional component utilization threshold

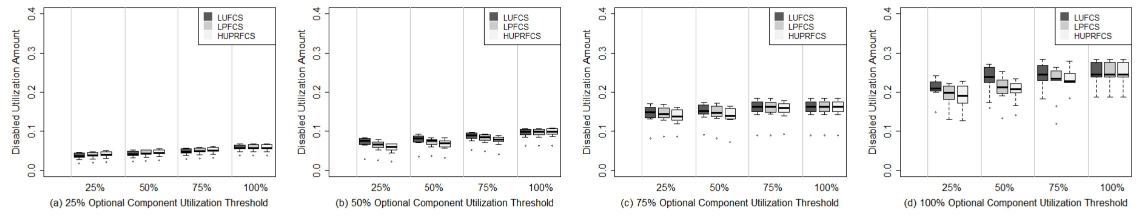


Figure 3.10: Disabled utilization amount comparison by varying percentage of connected components and optional component utilization threshold

served when more optional components are provided.

Although LUFCS with 100% optional components saves about 19% and 16% energy for approximate components and different components respectively, it is not recommended to set all components as optional since too much discount amount is offered. We will discuss policies selection considering the trade-offs in the Section 3.3.5.

Connected Components

After investigating the components without connections, we move to investigate connected components. As mentioned in Algorithm 3.2, in these cases, our proposed policies treat the connected components together and use their average utilization or discount to sort. Figure 3.6 shows the PCO, UBP, NUFCS and LUFCS for approximate components when varying the percentage of connected components (optional compo-

Table 3.7: Recommended policies for components without connections under different configurations

Components Design Pattern	Discount Constraint	Optional Component Utilization Threshold	Percentage of Optional Components	Recommend Policy	Algorithm Efficiency
Approximate Components	$\leq$ 5%	$\leq$ 100%	$\leq$ 100%	NUFCS	0.947 with 95% CI: (0.941, 0.953)
	>5%			LUFCS	0.91 with 95% CI: (0.887, 0.933)
Different Components	$\leq$ 100%	$\leq$ 50%	$\leq$ 100%	LUFCS	0.918 with 95% CI: (0.895, 0.941)
		50%-75%		LPFCS	0.913 with 95% CI: (0.891, 0.936)
		>75%		HUPRFCS	0.908 with 95% CI: (0.885, 0.931)

Table 3.8: Recommended policies for connected components under different configurations

Components Design Pattern	Discount Constraint	Percentage of Connected Components	Optional Component Utilization Threshold	Percentage of Optional Components	Recommended Policy	Algorithm Efficiency with 95% CI
Approximate Components	$\leq$ 5%	$\leq$ 100%	$\leq$ 100%	$\leq$ 100%	NUFCS	0.901 (0.852, 0.953)
	>5%		$\leq$ 100%		LUFCS	0.877 (0.852, 0.902)
Different Components	$\leq$ 100%	$\leq$ 50%	$\leq$ 50%	$\leq$ 100%	LUFCS	0.895 (0.859, 0.931)
			>50%		LPFCS	0.89 (0.858, 0.922)
		50%-75%	$\leq$ 50%		LPFCS	0.886 (0.855, 0.917)
			>50%		HUPRFCS	0.881 (0.856, 0.906)
		>75%	$\leq$ 100%		HUPRFCS	0.880 (0.85, 0.91)

nent utilization threshold and percentage of optional components are both fixed as 50%). Figure 3.6a shows that the connected components affects the NUFCS impressively. The energy consumption drops heavily in NUFCS when the percentage of connected components increases, i.e., from 9% to 21% reduction compared with UBP. While in LUFCS, the connected components do not affect its performance significantly. Although the energy consumption is also reduced when the percentage of connected components increases, energy consumption drops slowly from 14% to 21%. When 100% components are connected, NUFCS and LUFCS produce the same effects. As shown in Figure 3.6b, with the increase of connected components, discount amount increases fast from 10% to 23% in NUFCS while slowly in LUFCS from 17% to 23%. NUFCS and LUFCS both offer same discount amount when all the components are connected. For the cases that save more energy, like NUFCS or LUFCS with 100% connected components, Figure 3.6c and Figure 3.6d show that more utilization amount is disabled and more hosts are shutdown than baseline algorithms.

Figure 3.7 illustrates the comparison of LUFCS, LPFCS and HUPRFCS for different components when varying the percentage of connected components. Figure 3.7a shows that when connected components are larger than 75%, these policies do not result in significant differences, this is due to when the percentage of connected components increases, similar deactivated component lists are obtained although these components may be put into the list in different orders by these policies. Apparent differences for discount amount and disabled utilization amount are illustrated in Figure 3.7b and Figure 3.7c when connected components are less than 75%, like LUFCS reduces 2% to 5% energy than LPFCS and 5% to 10% energy than HUPRFCS, LUFCS offers 4% to 10% more discount than LPFCS and 9% to 15% more discount amount than HUPRFCS. Figure 3.7d shows that when components are connected, more hosts are shutdown than

UBP. In summary, our proposed multiple components selection policies works better under components with lower connected percentage up to 75%, which enables to provide multiple choices for service provider rather than providing same effects.

To evaluate the effects of combined parameters, we vary optional component utilization threshold and percentage of connected components together. We choose the optional component utilization threshold, as this parameter shows more significant effects than the percentage of optional component in energy and discount. Figures 3.8 to 3.10 demonstrate the energy consumption, discount amount and disabled utilization amount separately when varying these two parameters together. Each subfigure is with fixed optional component utilization threshold and variable percentage of connected components, for example, Figure 3.8a represents energy consumption when optional component utilization threshold is 25% and percentage of connected component is varied from 25% to 100%. Figure 3.8 shows that energy is reduced when connected components increases or larger optional component utilization threshold is given. For the compared policies, LUFCS, LPFCS and HUPRFCS show similar results when optional component utilization threshold is below 25% or percentage of connected components is above 75%. This is because when optional component utilization threshold is low, the disabled utilization is quite close for different policies, and higher percentage of connected components also contributes to deactivating the same list of components. For other cases that show statistically significant differences in energy consumption with *p-value* less than 0.05, like in Figure 3.7, the results are given as LUFCS $\leq$ LPFCS $\leq$ HUPRFCS. In these cases, Figure 3.9 and Figure 3.10 also show that LUFCS $>$ LPFCS $>$ HUPRFCS in discount amount and disabled utilization.

In conclusion, EEBA algorithm saves more energy than the VM consolidation approaches without brownout, like PCO and UBP. It is noticed that our experiments are mainly focused on optimizing servers energy consumption, so the network infrastructure energy consumption is not optimized. Since the component selection policies in brownout controller can be modeled into applications, like in [58], they are insensitive to network infrastructures.

Policy Selection Recommendation

To help make choices for component selection policies, we use Equation 3.9 to calculate their algorithm efficiency and summarize suitable policies under different configurations to achieve better energy efficiency. We consider energy consumption and discount with the same importance, so the α is set as 1. Table 3.7 shows the results for components without connections and Table 3.8 presents the results for the connected components.

To sum up, for components without connections, 1) when the components are approximate, NUFCS fits in the configurations when service provider allows maximum 5% discount and LUFCS performs better when more discount amount is allowed by service provider. 2) When the components are different, although the discount constraint is not as important as in the approximate components cases, the policies are picked out by other parameters, for instance, LUFCS achieves the best efficiency with less than 50% optional component utilization threshold, LPFCS overwhelms others with 50% to 75% optional component utilization threshold, HUPRFCS performs the best efficiency with more than 75% optional components utilization threshold.

For connected components, the suitable conditions are more complex: 1) when the components are approximate, NUFCS is recommended if the discount amount is limited under 5% and LUFCS is suggested if more than 5% discount amount is allowed; 2) when the components are different, recommended policy changes via different configurations. For example, when connected components are less than 50%, if optional component utilization threshold is less than 50%, LUFCS is recommended; if optional component utilization threshold is larger than 50%, LPFCS is recommended. When connected components are between 50% and 75, LPFCS is recommended for optional component utilization threshold that is not larger than 50%, HUPRFCS is recommended for optional component utilization threshold larger than 50%. When more than 75% components are connected, any policy achieves quite close results, HUPRFCS is a choice.

3.6 Summary

In this chapter, we introduced the brownout enabled system model by considering application components, which are either mandatory or optional. In the model, the brownout controller can deactivate the optional components to reduce data center energy consumption while offering discount to users. We also proposed a brownout enabled algorithm to determine when to use brownout and how much utilization on a host is reduced. Then we presented a number of policies to select components and investigate their effects on energy consumption and offered discount.

In our experiments, we considered different configurations, such as components without connections, connected components, approximate components, different components and etc. The results showed that these proposed policies save more energy than the baselines PCO and UBP. The comparison of proposed policies demonstrates that these policies fit in different configurations. Considering the discount amount offered by a service provider, NUFCS is recommended when a small amount of discount (like less than 5%) is offered, as it can reduce maximum 7% energy consumption in contrast to UBP. When more discount amount (like more than 5%) is allowed by service provider, other multiple components selection policies are recommended, for example, compared with UBP, HUPRFCS saves more than 20% energy with 10% to 15% discount amount.

However, this chapter adopts the algorithm based on heuristic algorithms, which is fast while the optimal results may not be obtained. This means the trade-offs between energy and discount can be improved. In the next chapter, we apply the algorithm based on Markov Decision Process, which belongs to the meta-heuristic algorithm. The proposed algorithm searches a large solution space and aims to find more possible combinations of deactivated components and achieve better trade-offs.

Chapter 4

Energy Efficient Scheduling of Application Components via Brownout and Approximate Markov Decision Process

Unexpected loads in Cloud data centers may trigger overloaded situation and performance degradation. Brownout has been proved to be a promising approach to relieve the overloads through deactivating application non-mandatory components or microservices temporarily. Moreover, brownout has been applied to reduce data center energy consumption. It shows that there are trade-offs between energy saving and discount offered to users (revenue loss) when one or more services are not provided temporarily. In this chapter, we propose a brownout-based approximate Markov Decision Process approach to improve the aforementioned trade-offs. The results based on real trace demonstrate that our approach saves 20% energy consumption than VM consolidation approach. Compared with existing energy-efficient brownout approach, the proposed approach reduces the discount amount given to users while saving similar energy consumption.

4.1 Introduction

BROWNOUT has shown its effectiveness in relieving overloads and improving energy efficiency for cloud computing system. When brownout is used to optimize energy, discount can be the trade-off. In our scenario, the meaning of discount is not limited to the discount offered to users. Additionally, it can also be modeled as the revenue loss of service providers (i.e. SaaS service providers) that they charge the lower price for

This chapter is derived from:
•**Minxian Xu** and Rajkumar Buyya, "Energy Efficient Scheduling of Application Components via Brownout and Approximate Markov Decision Process," *in Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC)*, LNCS, Springer-Verlag Press, Berlin, Germany), Malaga, Spain, November 13-16, 2017.

services under brownout. For example, in an online shopping system, the recommendation engine helps the service provider to improve their revenue by recommending similar products. If the recommendation engine is deactivated, the service provider is unable to obtain the revenue from the recommendation engine.

In this chapter, we consider to apply meta-heuristic algorithm based on Markov Decision Process to improve algorithm performance. We consider component-level control in our system model, which could also be applied to container or microservices architecture. We model the application components as either mandatory or optional, and if required, optional components can be deactivated. By deactivating the optional components selectively and dynamically, the application utilization is reduced to save total energy consumption. While under market scenario, service provider may provide discount for users as one or more services are deactivated.

The key **contributions** of this chapter are:

- Our approach considers the trade-offs between saved energy and the discount that is given to a user if components or microservices are deactivated;
- We propose an efficient algorithm based on brownout and approximate Markov Decision Process that considers the aforementioned trade-offs and achieves better trade-offs than baselines.

The remainder of this chapter is organized as follows: after discussing the related work in Section 4.2, we present the brownout system model and problem statement in Section 4.3. Section 4.4 introduces our proposed brownout-based Markov Decision Process approach, and Section 4.5 demonstrates the experimental results of our proposed approach. The summary is concluded in Section 4.6.

4.2 Related Work

A large body of literature has focused on reducing energy consumption in cloud data centers, and the dominant categories for solving this problem are VM consolidation and Dynamic Voltage Frequency Scaling (DVFS).

VM consolidation is viewed as an act of combining into an integral whole, which saves energy by allocating work among fewer machines and turning off unused machines [123]. Using this approach, VMs allocated to underutilized hosts are consolidated to other servers and the remaining hosts are transformed into low power mode. Mastroianni et al. [105] presented a self-organizing and adaptive approach for consolidation of VMs CPU and RAM resource, which is driven by probabilistic processes and local information. Corradi et al. [47] considered VM consolidation in a more practical viewpoint related to power, CPU and networking resource sharing and tested VM consolidation in OpenStack, which shows VM consolidation is a feasible solution to lessen energy consumption. Theja et al. [141] proposed an adaptive genetic algorithm based on VM consolidation for QoS and energy oriented Clouds. Monil et al. [111] developed a fuzzy logic and heuristic-based VM migration control approach to achieve the trade-off between energy and QoS. Fuzzy VM selection method is applied to select VM from overloaded host and migrate. Hosseinimotlagh et al. [77] presented a VM scheduling algorithm based on the unsurpassed utilization level to achieve optimized energy consumption while meeting a QoS constraint.

The DVFS technique introduces a trade-off between computing performance and energy consumed by the server. The DVFS technique lowers the frequency and voltage when the processor is lightly loaded, and utilizes maximum frequency and voltage when the processor is heavily loaded. Kim et al. [83] proposed several power-aware VM schemes based on DVFS for real-time services. Hanumaiah et al. [72] introduced a solution that considers DVFS, thread migration and active cooling to control the cores to maximize overall energy efficiency.

Most of the proposed brownout approaches in Cloud scenarios focused on handling overloads or overbooking rather than energy efficiency perspective. Klein et al. [85] firstly borrowed the approach of brownout and applied it to cloud applications, aiming to design more robust applications under unpredictable loads. Tomas et al. [142] used brownout along with overbooking to ensure graceful degradation during load spikes and avoid overload. In a brownout-compliant application or service, the optional parts are identified by developers, and a control knob called **dimmer** that controls these optional parts is also introduced. The dimmer value represents a certain probability given by a control variable and shows how often these optional parts are executed. Moreover, a brownout controller is also required to adjust the dimmer value.

Markov Decision Process (MDP) is a discrete time stochastic optimization approach and provides a way to solve the multiple state probabilistic decision-making problem, which has been adopted to solve resource management problems in Cloud scenarios. Toosi et al. [144] used finite MDP for requests admission control in Clouds, while their objective is maximizing revenues rather than reducing power consumption. Han et al. [71] applied MDP to determine VM migration for minimizing energy consumption, while our work is adopting MDP to determine the deactivation of application components.

In our previous chapter, several heuristic policies were proposed to find the components that should be deactivated and investigated the trade-offs between energy and discount. In this chapter, we adopt approximate MDP to improve the aforementioned trade-offs.

4.3 System Model and Problem Definition

4.3.1 System Model

Our system model is presented in Figure 4.1 and it consists of the following entities:

Users: Users submit service requests to cloud data centers. The users entity contains user information and requested applications (services).

Applications: The applications provide different services for users and are consisted of a set of components, which are identified as mandatory or optional. Different from Figure 3.1, we don't consider the connected components in this chapter.

Mandatory component: The mandatory component keeps running all the time when the application is launched.

Optional component: The optional component can be set as activated or deactivated according to the system status. These components have parameters like utilization $u(App_c)$ and discount $d(App_c)$. Utilization indicates the amount of utilization, and discount represents the amount of discount that is offered to the users (or revenue loss of service provider). The operations of optional components are controlled by the **brownout controller**, which makes decisions based on the system overloaded status and brownout algorithm.

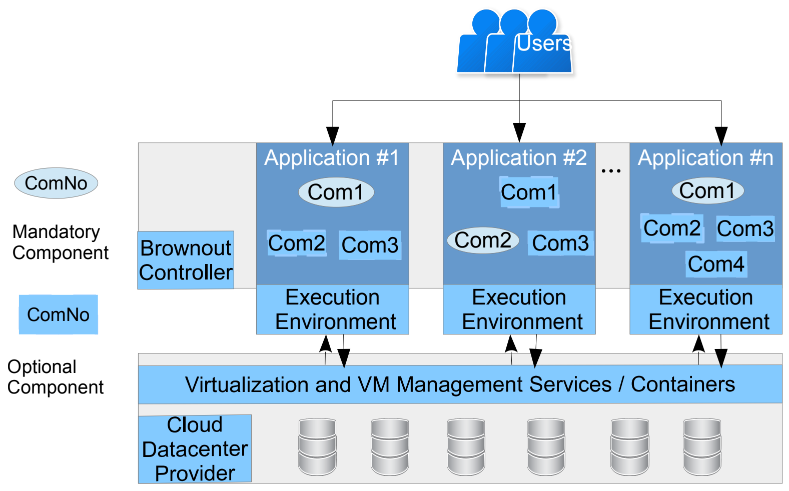


Figure 4.1: System Model with Brownout

To adapt the dimmer to our model, different from the dimmer in [85] that requires a dimmer per application, our dimmer is only applied to the applications with optional components. Rather than response time, another adaptation is that our dimmer value is computed based on the number of overloaded hosts and adapts to the severity of overloaded events (more details are presented in Section 4.4.1).

Cloud Providers: Cloud providers offer physical resources to meet service demands, which host a set of VMs or containers to run applications.

4.3.2 Power Model

We adopt the servers power model derived from [155]. The power of server i is $P_i(t)$ that is dominated by the CPU utilization:

$$P_i(t) = \begin{cases} P_i^{idle} + \sum_{j=1}^{N_i} u(VM_{i,j}(t)) \times P_i^{dynamic} & , N_i > 0 \\ 0 & , N_i = 0 \end{cases} \tag{4.1}$$

$P_i(t)$ is composed of idle power and dynamic power. The idle power is regarded as constant and the dynamic power is linear to the total CPU utilization of all the VMs on the server [155]. If no VM is hosted on a server, the server is turned off to save power. $VM_{i,j}$ refers to the jth VM on server i, N_i means the number of VMs assigned to server i. And $u(VM_{i,j}(t))$ refers to the VM utilization at time interval t, which is represented as:

$$u(VM_{i,j}(t)) = \sum_{c=1}^{C_j} u(App_c) \tag{4.2}$$

where C_j is the number of application components on VM, and $u(App_c)$ is the utilization of application component c when it is activated.

Then the total energy consumption during time interval t, with M servers is:

$$E(t) = \sum_{i=1}^{M} \int_{t-1}^{t} P_i(t)dt \tag{4.3}$$

Notes: In our power model, we assume that the time required to turn on/off hosts (including the time to deactivate and activate components) is less than a scheduling time interval (like 5 minutes). When the host is turned off/on, the host is assumed to be consuming the idle power.

4.3.3 Discount Amount

As introduced in Section 4.1, the meaning of discount could be either the discount offered to users or the revenue loss of service providers that they charge the lower price for services under brownout. In this chapter, we note them as discount.

The total discount amount at time interval t is modeled as the sum of discount of all deactivated application components at t:

$$D(t) = \sum_{i=1}^{M} \sum_{j=1}^{N_i} d(VM_{i,j}(t)) \tag{4.4}$$

where $D(t)$ is the total discount amount at t that obtained from all VMs on hosts, N_i is the number of VMs assigned to server i, M is the number of servers. The individual discount $d(VM_{i,j}(t))$ is the sum of discount amount of deactivated application components $d(App_c)$ of $VM_{i,j}$, which is shown in Equation 4.5:

$$d(VM_{i,j}) = \sum_{c=1}^{C_j} d(App_c) \tag{4.5}$$

where C_j is the number of application components hosted on VM_j, and only the deactivated components are counted.

4.3.4 Problem Definition

Let $Q(t) \in \mathbb{Q}$, where $\mathbb{Q} = \eta_1, \ldots, \eta_{|\mathbb{Q}|}$, $\eta_i \in \mathbb{Q}$. The $Q(t)$ is a combination of two vectors: energy consumption vector $E(t)$ and discount amount vector $D(t)$, representing the possible energy consumption and discount amount at different system states. Let $C(t)$ to be all the application component states at t, we have

Definition 1 *The system state at time interval t can be specified as:*

$$S(t) \triangleq [Q(t), C(t)] \tag{4.6}$$

The system state $S(t)$ contains the energy consumption and discount amount as well as their corresponding application components states .

At each time interval, we calculate the state information as:

$$g(t) = E(t) + \lambda D(t) \tag{4.7}$$

where λ is the weight of discount. The higher λ implicates that more weights are given to the discount amount. In the whole scheduling period T under policy π, our optimization objective is:

$$\min_{\pi} \qquad g(\pi) = \sum_{t=0}^{T} [E(t) + \lambda D(t)] \tag{4.8}$$

4.4 Proposed Approach

4.4.1 Approximate Markov Decision Process Model

To adopt the Markov model, we assume that the workload satisfies the Markov property, which means the state transitions caused by workloads are memoryless. Our experiments are conducted with PlanetLab workload, which has been validated to satisfy Markov chain property [33]. In our model, we assume that the probability of application components to transfer their states at the next time period only depends on the workloads of the current time period and independent on earlier states. We formulate our problem as finite horizon MDP that we investigate a fixed length of time.

Then we can solve our objective function by using Bellman equation [31]:

$$V^*(S_i) = \arg\min_{\gamma \in \mathbb{R}} [g(S_i) + \sum_{S_j \in S} Pr[S_j | S_i, \gamma] V^*(S_j)] \tag{4.9}$$

$g(S_i)$ is the instant cost under system state S_i, and $V^*(S_i)$ is the expected energy consumption and discount obtained from S_j to S_i. We also denote $\gamma(t) \triangleq [\gamma_1(t), \ldots, \gamma_n(t)] \in \mathbb{R}$ as the operations (activation or deactivation actions) for application components. $V^*(S_i)$ can be found by iteratively obtaining minimum energy consumption and discount until convergence.

Let $\hat{p_{i,j}}$ denote the estimated transition probability that the application component changes its state. The transition probability is computed as:

$$\hat{p}_{i,j} = \sqrt{\frac{\hat{M}}{M}} \times Pr(\frac{u(App_c)}{d(App_c)} = z_C) \tag{4.10}$$

$Pr(\frac{u(App_c)}{d(App_c)} = z_C)$ is the probability that the ratio of component utilization and discount $\frac{u(App_c)}{d(App_c)}$ falls into category z_C. We divide the probability into C (the maximum number of components on a VM) categories. For all the components with the probability falls into the same category, we apply the same operation. To avoid the curse of dimension, noted by [71], we adopt key states to reduce state space. With key states, the component states on a VM is reduced to the maximum number of components on a VM as $|C|$. $\hat{M}$ is the estimated number of overloaded hosts, which is calculated based on a sliding window [32]. The advantage of sliding window is to give more weights to the values of recent time intervals. Let L_w to be the window size, and $N(t)$ to be the number of overloaded hosts at t, we estimate $\hat{M}$ as:

$$\hat{M}(L_w) = \frac{1}{L_w} \sum_{t=0}^{L_w - 1} N(t) \tag{4.11}$$

We denote the states as key states S_k as described above. With proof in [71], $\forall S_i \in S_k$ for all the VMs, the equivalent Bellman's equation in Equation 4.9 can be approximately formulated as:

$$V^*(S_i) \approx \sum_{m=1}^{M} \sum_{n=1}^{N_m} (g(S_i) + \arg\min_{\gamma_n \in \mathbb{R}_n} \{ \sum_{S_j \in S_k} Pr[S_j | S_i, \gamma_n] \widetilde{V_n^*}(S_j) \}) \tag{4.12}$$

The state spaces thus are reduced to polynomial with linear approximation. The M is the number of hosts and N_m is the number of VM assigned to server m.

4.4.2 Brownout Algorithm based on Markov Decision Process (BMDP)

Our novel brownout algorithm is embedded within a VM placement and consolidation algorithm. We adopt the VM placement and consolidation algorithm (PCO) proposed in [33], which is also one of our baselines in Section 4.5.

The PCO algorithm is a heuristic to reduce energy consumption through VM consolidation. In the initial VM placement phase, PCO sorts all the VMs in decreasing order by their current CPU utilization and allocates each VM to the host that increases the least power consumption due to this allocation. In the VM consolidation phase, PCO optimizes VM placement by separately picking VMs from over-utilized and under-utilized hosts to migrate, and finding new placements for them. After migration, the over-utilized hosts are not overloaded any more and the under-utilized hosts are switched to sleep mode.

Our brownout algorithm based on approximate Markov Decision Process is shown in Algorithm 4.1 and includes 6 steps:

1) System initialization (lines 1-2): Initializing the system configurations, including overloaded threshold TP, dimmer value θ_t, vector $\mathbb{Q}$ that contains the $D(t)$ and $E(t)$ information, as well as objective states $\mathbb{S}_d$, and applying VM placement algorithm in PCO to initialize VM placement.

2) Estimating transition probability of each application component (lines 3-14): At each time interval, the algorithm firstly estimates the number of overloaded hosts. The dimmer value is computed as $\sqrt{\frac{\hat{M}}{M}}$, which is adaptive to the number of overloaded hosts. If no host is overloaded, the value is 0 and no component is deactivated. If there are overloaded hosts, the transition probabilities of application components are computed using Equation 4.10.

3) Finding the states that minimize the objective function (lines 15-17): Traversing all the key states by value iteration according to Equation 4.12, where $D^{'}(t)$ and $E^{'}(t)$ are the temporary values at the current state.

4) Updating system information (lines 18-20): The algorithm updates the obtained energy consumption and discount values if $g(t)$ in Equation 4.7 is reduced, and records

Algorithm 4.1: Brownout based Markov Decision Process Algorithm (BMDP)

```
Input  : host list $hl$ with size $M$, VM list, application components information, overloaded power threshold $TP$,
         dimmer value $\theta_t$ at time $t$, destination states $S_d(t)$, energy consumption $E(t)$ and discount amount
         $D(t)$ in $\mathbb{Q}$
Output: total energy consumption, discount amount
1  $TP \leftarrow 0.8; \theta_t \leftarrow 0; \forall E(t), \forall D(t) \in \mathbb{Q} \leftarrow max; S_d(t) \in \mathbb{S}_d \leftarrow NULL$
2  use PCO algorithm to initialize VMs placement
3  while true do
4      for $t \leftarrow 0$ to $T$ do
5          $\theta_t \leftarrow = \sqrt{\frac{\hat{M}_t}{M}}$
6          forall $h_i$ in $hl$ do
7              if $h_i$ is overloaded then
8                  forall $VM_{i,j}$ on $h_i$ do
9                      forall $App_c$ on $VM_{i,j}$ do
10                         $Pr(App_c) \leftarrow \theta_t \times Pr(\frac{u(App_c)}{d(App_c)} = z_C)$
11                     end
12                 end
13             end
14         end
15         forall $S_j(t) \in S_k(t)$ do
16             $V^*(S_i) = \sum_{m=1}^{m=M} \sum_{n=1}^{n=N_m} (g(S_i) + \min_{\gamma_n \in \mathbb{R}_n} \{\sum_{S_j \in S_k} Pr[S_j|S_i, \gamma_n] \widetilde{V_n^*}(S_j)\})$
17             $g(t) = E^{'}(t) + \lambda D^{'}(t)$
18             if $g(t) < E(t) + \lambda D(t)$ then
19                 $E(t) \leftarrow E^{'}(t)\ ; D(t) \leftarrow D^{'}(t)\ ; S_d(t) \leftarrow S_j(t)$
20             end
21             deactivate the selected components to achieve state $S_d(t)$
22         end
23     end
24     use VM consolidation in PCO algorithm to optimize VM placement
25 end
```

the optimized states. The current states are substituted by the state with a lower $g(t)$.

5) Deactivating the selected components (line 22): The brownout controller deactivates the selected components to achieve objective states.

6) Optimize VMs placement (line 24): The algorithm uses the VM consolidation approach in PCO to optimize VM placement via VM consolidations.

The complexity of the BMDP algorithm at each time interval is consisted of the brownout part and VM consolidation part. The complexity of the transition probability computation is $O(C \cdot N \cdot M)$, where C is the maximum number of components in all applications, N is the maximum number of VMs on all the hosts and M is the number of hosts. With the key states, the space state of the MDP in brownout part is $O(C \cdot N \cdot M)$. According to Equation 4.12, the actions are reduced to $O(C \cdot N \cdot M)$, so the overall MDP complexity is $O(C^2 \cdot N^2 \cdot M^2)$. The complexity of the PCO part is $O(2M)$ as analyzed in [33]. Therefore, the overall complexity is $O(C \cdot M \cdot N + C^2 \cdot N^2 \cdot M^2 + 2M)$ or equally $O(C^2 \cdot N^2 \cdot M^2)$.

Table 4.1: Host / VM Types and capacity

Name	CPU	Cores	Memory	Bandwidth	Storage
Host Type 1	1.86 GHz	2	4 GB	1 Gbit/s	100 GB
Host Type 2	2.66 GHz	2	4 GB	1 Gbit/s	100 GB
VM Type 1	2.5 GHz	1	870 MB	100 Mbit/s	1 GB
VM?Type 2	2.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 3	1.0 GHz	1	1740 MB	100 Mbit/s	1 GB
VM Type 4	0.5 GHz	1	613 MB	100 Mbit/s	1 GB

4.5 Performance Evaluation

4.5.1 Methodology

We use the CloudSim framework [40] to simulate a cloud data center. The data center contains two types of hosts and four types of VMs that are modeled based on current offerings in EC2 as shown in Table 4.1. The power models of the adopted hosts are derived from IBM System x3550 M3 with CPU Intel Xeon X5670 and X5675 [1] . We set the time required to turn on/off hosts as 0.5 minute.

We implemented an application with optional components, and each component has its corresponding CPU utilization and discount amount. The components are uniformly distributed on VMs.

We adopt the realistic workload trace from more than 1000 PlanetLab VMs [122] to create an overloaded environment [32]. Our experiments are simulated under one-day scheduling period and repeated for 10 different days. The brownout is invoked every 5 minutes (one time interval) if hosts are overloaded. The sliding window size L_w in Equation 4.11 to estimate the number of overloaded hosts is set as 12 windows (one hour).

The CPU resource is measured with capacity of running instructions. Assuming that the application workload occupies 85% resource on a VM and the VM has 1000 million instructions per second (MIPS) computation capacity, then it represents the application constantly requires $0.85 \times 1000 = 850$ MI per second in the 5 minutes time interval.

We use three baseline algorithms for comparison as below:

1) VM Placement and Consolidation algorithm (PCO) [33]: the algorithm has been described at the beginning of Section 4.2.

2) Utilization-based Probabilistic VM consolidation algorithm (UBP) [43]: for VM initial placement, UBP adopts the same approach as PCO, which sorts all the VMs in decreasing order based on their utilization and allocating each VM to the host that increases the least power consumption. For VM consolidation, UBP applies a probabilistic method [105] to select VMs from overloaded host. The probabilistic method calculates the migration probability $f_m(u)$ based on host utilization u as : $f_m(u) = (1 - \frac{u-1}{1-T_h})^{\alpha}$, where T_h is the upper threshold for detecting overloads and α is a constant to adjust probability.

3) Brownout algorithm with Highest Utilization and Price Ratio First Component Selection Algorithm (HUPRFCS) from Chapter 3: it is a brownout-based heuristic algorithm. This algorithm deactivates the application components from the one with the highest $\frac{u(App_)}{d(App_c)}$ to the others with lower $\frac{u(App_c)}{d(App_c)}$ until the deactivated components obtain the expected utilization reduction, which is a deterministic algorithm. HUPRFCS is an efficient approach to reduce energy consumption under discount amount constraints.

To evaluate algorithms' performance, we mainly explore two parameters:

1) Overloaded threshold: It identifies the CPU utilization threshold that determines the overloaded hosts, and it is varied from 80% to 95% in increments of 5%. We adopt this parameter since both [33] and [105] have shown that it influences energy consumption.

2) Percentage of optional utilization in an application: It shows how much utilization in application is optional and can be deactivated. It is varied from 25% to 100% in increments of 25%. An application with 100% optional utilization represents that the application components or microservices are self-contained and each of them is allowed to be disabled temporarily (not disabling all the components at the same time), such as a stateless online document processing application. We assume the application maximum discount is identical to the percentage of optional utilization, for example, 50% optional utilization in an application comes along with 50% discount amount.

We assume that the optional components utilization $u(App_c)$ and discount $d(App_c)$ conform normal distribution $u(App_c) \sim N(\mu, \sigma^2)$, $d(App_c) \sim N(\mu, \sigma^2)$, the μ is the mean utilization of component utilization or discount, which is computed as the percentage of optional utilization (or discount amount) divided by the number of optional components. The σ^2 is the standard deviation of optional components utilization or discount. In our experiments, we consider both optional component utilization standard deviation and discount standard deviation are less than 0.1, which represents that the optional components are designed to have balanced utilization and discount.

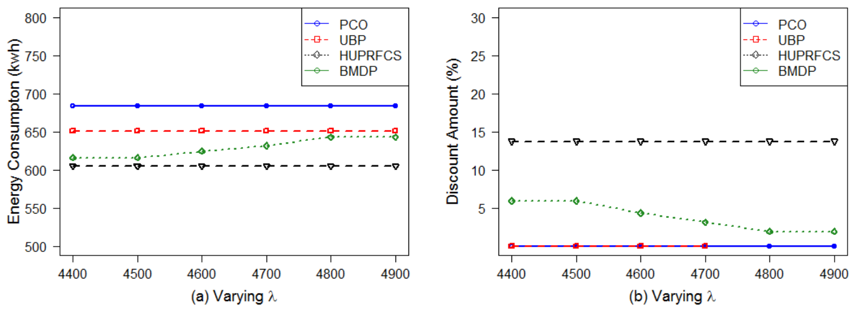


Figure 4.2: Comparison with different λ. The parameter λ is the weight of discount.

4.5.2 Results

4.5.2.1 Comparison with different λ

To investigate the impacts of different discount weights in Equation 4.7, we conduct a series of experiments with different λ. In these evaluations, the hosts number and VMs number are set to 200 and 400 respectively, the overloaded threshold is set to 85% and the percentage of optional utilization is set to 50%. Figure 4.2 indicates that energy consumption increases and discount amount decreases when λ increases. The reason lies in that larger λ will guide our algorithm to find the states that offer less discount. From the results, we notice that when λ value is less than 4500, BMDP saves more energy than UBP and PCO, and in comparison to HUPRFCS, BMDP has similar energy consumption and reduces significant discount amount.

In the following evaluations, we set λ to a small value (i.e. λ=100) so that the energy consumption of BMDP is below two baselines (PCO and UBP) and close to HUPRFCS. Additionally, with this λ value, the discount of BMDP is less than the discount produced by HUPRFCS.

4.5.2.2 Comparison under varied overloaded thresholds

The performance evaluated under different overloaded thresholds is shown in Figure 4.3. Other parameters are configured as same as in Section 4.5.2.1. In Figure 4.3a, we observe that the energy consumption of all the algorithms are reduced when the overloaded threshold increases, for example, PCO-80% has 699.6 kWh with 95% Confidence Interval (CI) (682.6, 716.6) and reduces it to 649.9 kWh with 95% CI: (635.8, 664.1) in PCO-95%; BMDP-80% has 607.8 kWh with 95% CI: (598.1, 617.4) and saves it as 558.4 kWh with

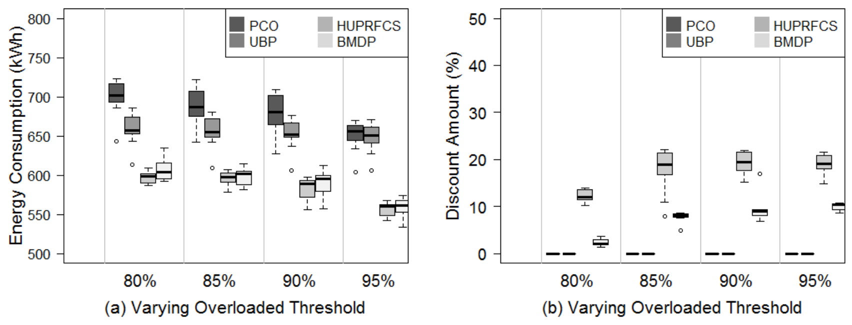


Figure 4.3: Varying overloaded threshold

Table 4.2: Paired t-tests with 95% CIs for comparing energy consumption by HUPRFCS and BMDP under different overloaded thresholds

Algorithm 1 (kWh)	Algorithm 2 (kWh)	Difference (kWh)	*p-value*
HUPRFCS-80% (598.01)	BMDP-80% (607.78)	-9.77 (-15.14, -4.39)	0.0026
HUPRFCS-85% (595.87)	BMDP-85% (599.24)	3.37 (-0.77, 7.52)	0.099
HUPRFCS-90% (581.91)	BMDP-90% (587.97)	-6.05 (-9.41 -2.69)	0.0027
HUPRFCS-95% (557.03)	BMDP-95% (558.41)	-1.38 (-5.36, 2.6)	0.45

95% CI: (549.6, 567.2) in BMDP-95%. The reason lies in that higher overloaded thresholds allow more VMs to be packed on the same host, so that more hosts are shutdown. When overloaded thresholds are between 80% to 90%, UBP reduces around 5% energy consumption compared to PCO, while HUPRFCS and BMDP save about 14-16% more energy consumption than PCO. When the overloaded threshold is 95%, PCO and UBP achieve close energy consumption, while HUPRFCS and BMDP still reduce around 16% energy compared with them.

As the energy consumption of HUPRFCS and BMDP are quite close, we conduct paired t-tests for HUPRFCS and BMDP as shown in Table 4.2. We notice that the differences between them are less than 2%, and when the overloaded thresholds are 85% and 95%, the *p-values* are 0.09 and 0.45 respectively, which indicates weak evidence to prove that they are different.

Comparing the discount amount, Figure 4.3b shows that there is no discount offered in PCO and UBP, but HUPRFCS offers 11% to 20% discount and BMDP reduces it to 3% to 11% as the trade-off due to components deactivation. This is because, based on heuristics, HUPRFCS quickly finds the components with higher utilization and discount ratio, while BMDP steps further based on MDP to optimize the component selection.

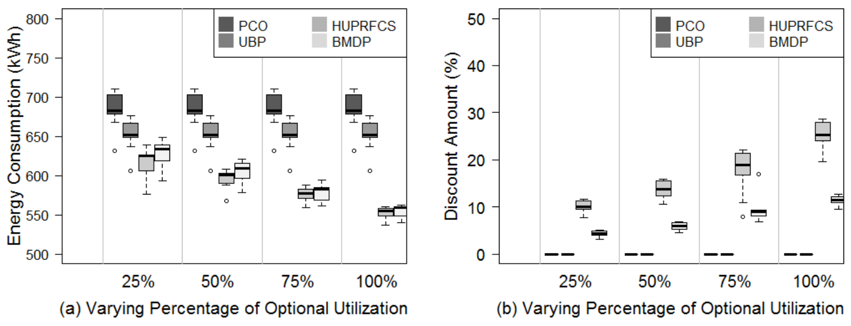


Figure 4.4: Varying percentage of optional utilization

4.5.2.3 Comparison under Varied Percentage of Optional Utilization

In Figure 4.4, we compare the algorithms with different percentages of optional utilization. Other parameters are set the same as those in Section 4.5.2.1. As shown in Figure 4.4a, for PCO and UBP, their energy consumption are not influenced by different percentage of optional utilization. PCO has 684 kWh with 95% CI: (667.4, 700.6), and UBP has reduced 4.7% to 651.9 with 95% CI: (637.3, 666.5). Compared with PCO, HUPRFCS-25% reduces 11% energy to 605kWh with 95% CI: (596.6, 613.4), and BMDP-25% reduces 9% energy to 615.9 kWh with 95% CI: (605.9, 625.8). When the percentage of optional utilization increases, the more energy consumption is saved by HUPRFCS and BMDP. For instance, HUPRFCS-100% and BMDP-100% achieve around 20% energy saving as 556.9kWh with 95% CI: (550.9, 562.3) and 551.6kWh with 95% CI: (545.8, 557.4) respectively. The reason is that higher percentage of optional percentage allows more utilization to be reduced. For the discount amount comparison in Figure 4.4b, it shows that HUPRFCS offers 10% to 25% discount amount as trade-offs, while BMDP only offers 3% to 10% discount amount. Figure 4.4c demonstrates that HUPRFCS provides about 3% more disabled utilization amount than BMDP. In the comparison of the number of shutdown hosts shown in Figure 4.4d, HUPRFCS and BMDP shut down about 18-20 more host than PCO and 3-5 more hosts than UBP.

Because the energy consumption of HUPRFCS and BMDP are quite close, we conduct the paired t-test for HUPRFCS and BMDP as illustrated in Table 4.3. When the percentage of optional utilization are 75% and 100%, the *p-values* are 0.099 and 0.057, which indicates weak evidence to prove that they are different. And with other percentage of optional utilization, the energy consumption differences are less than 2%.

Table 4.3: Paired t-tests with 95% CIs for comparing energy consumption by HUPRFCS and BMDP under different percentage of optional utilization

Algorithm 1 (kWh)	**Algorithm 2 (kWh)**	**Difference (kWh)**	***p-value***
HUPRFCS-25% (617.57)	BMDP-25% (628.10)	-10.52 (-12.52, -8.52)	0.00082
HUPRFCS-50% (595.0)	BMDP-50% (605.88)	-10.88 (-15.26, -6.5)	0.00032
HUPRFCS-75% (575.87)	BMDP-75% (579.24)	-3.37 (-7.52 -0.78)	0.099
HUPRFCS-100% (551.56)	BMDP-100% (556.59)	-3.12 (-5.08, -1.16)	0.0057

4.6 Summary

In this chapter, we introduced the system model of brownout by deactivating optional components in applications or microservices temporarily. In the model, the brownout controller can deactivate the optional components or microservices to deal with overloads and reduce data center energy consumption while offering discount to users. We also proposed an algorithm based on brownout and approximate Markov Decision Process namely BMDP, to find the components should be deactivated. The simulations based on real trace showed that BMDP reduces 20% energy consumption than non-brownout baselines and saves discount amount than brownout baseline.

However, the evaluation of our brownout-based approaches is under simulations. There are some gaps between simulations and real testbeds, e.g. network traffics. In the next chapter, we present a brownout-based approach developed under real infrastructure, which validates the feasibility and scalability of brownout-based approach to reduce energy consumption while ensuring the quality of service.

Chapter 5

Integrated Approach for Managing Energy in Container-based Clouds

Energy consumption of Cloud data centers has been a major concern of many researchers, and one of the reasons for huge energy consumption of Clouds lies in the inefficient utilization of computing resources. Container-based clouds provide an approach to utilize resources in a more efficient manner. Besides energy consumption, another challenge of data centers is unexpected loads, which leads to overloads and performance degradation. In this chapter, we propose an integrated approach to manage energy consumption and brownout in container-based cloud data centers. We also evaluate our proposed scheduling policies with real traces in a prototype system. The results show that our approach reduces about 40%, 20% and 10% energy than the approach without power-saving techniques, brownout-overbooking approach and auto-scaling approach respectively while ensuring Quality of Service.

5.1 Introduction

DATA centers are required to offer resources while satisfying Quality of Service (QoS), as well as reduce energy consumption. Reducing energy consumption is a challenging objective as applications and data are growing fast and complex [96]. Normally, the applications and data are required to be processed within the required time, thus, large and powerful servers are required to offer services. To ensure the sustainability of future growth of data centers, cloud data centers must be designed to efficiently utilize the resources of infrastructure and minimize energy consumption. To address this problem, the concept of green cloud is proposed, which aims to manage cloud data

This chapter is derived from:
•**Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "iBrownout: An Integrated Approach for Managing Energy and Brownout in Container-based Clouds," *IEEE Transactions on Sustainable Computing (T-SUSC)*, Volume 4, Number 1, Pages: 53-66, ISSN: 2377-3782, IEEE Computer Society Press, USA, Jan-Mar 2019.

Table 5.1: Comparison of focus of related work and our work

Approach	Technique			Optimization Objective			Management Unit		Experiments Platform	
	VM Consolidation	DVFS	Brownout	Energy Consumption	SLA/QoS	Overloads	VMs	Containers	Simulation	Real Testbed
Beloglazov et al. [32]	✓	×	×	✓	✓	✓	✓	×	✓	×
Beloglazov et al. [34]	✓	×	×	✓	✓	×	✓	×	✓	×
Chen et al. [43]	✓	×	×	✓	✓	×	✓	×	✓	×
Han et al. [71]	✓	×	×	✓	✓	✓	✓	×	✓	×
Mastroianni et al. [105]	✓	×	×	✓	×	✓	✓	×	×	✓
Zheng et al. [155]	✓	×	×	✓	✓	×	✓	×	×	✓
Ferdaus et al. [62]	✓	×	×	✓	✓	×	✓	×	×	✓
Kim et al. [83]	×	✓	×	✓	✓	×	×	×	✓	×
Pietri et al. [125]	×	✓	×	✓	✓	×	×	×	✓	×
Teng et al. [140]	✓	✓	×	✓	✓	×	×	×	✓	✓
Klein et al. [85]	×	×	✓	×	✓	✓	×	×	×	✓
Tomas et al. [142]	×	×	✓	×	✓	✓	×	×	×	✓
Wang et al. [148]	✓	×	×	✓	✓	×	✓	×	×	✓
iBrownout	×	×	✓	✓	✓	✓	×	✓	×	✓

centers in an energy efficient manner [33].

In this chapter, we propose a brownout prototype system based on containers to reduce data center energy while ensuring Quality of Service. The main **contributions** of our work are as follows:

- Proposed an effective architecture that enables brownout paradigm to manage the container-based environment, which enables fine-grained control on containers;

- Presented several scheduling policies for managing microservices or containers to achieve power saving and QoS constraints;

- Implemented a prototype system and carried out the evaluation in INRIA Grid'5000 testbed using resources from Lyon cluster for Wikipedia web workload.

The rest of this chapter is organized as: Section 5.2 discusses the related work, followed by scenarios that brownout can be applied and the challenges for using brownout presented in Section 5.3. Section 5.4 and Section 5.5 introduce the architecture that enables brownout to manage the microservices or application components and models respectively. Scheduling policies for determining the activation and deactivation of microservices are presented in Section 5.6. In Section 5.7, we present our experiments environment and evaluate the performance of different scheduling policies. Conclusions and future directions are given in Section 5.8.

5.2 Related Work

A recent report suggests that U.S. data center will consume 140 billion kWh of electricity annually in the next four years by 2020 [29], which equals to the annual output of about 50 brown power plants and translates to higher carbon emissions. To decrease operational costs and environmental impact, numerous state-of-the-art research has been conducted to reduce data center energy consumption. The main categories for handling this energy efficient problem are VM consolidation and Dynamic Voltage Frequency Scaling (DVFS).

VM consolidation minimizes energy consumption by allocating tasks among fewer machines and turning the unused machines into low-power mode or power-off state. To reduce the number of active machines, the VMs hosted on underutilized machines are consolidated to other machines and the underutilized machines are transformed into low-power mode. Beloglazov et al. [32] proposed several VM consolidation algorithms to save data center energy consumption. The VM consolidation process is modeled as a bin-packing problem, where VMs are regarded as items and hosts are regarded as bins. The objective of these VM consolidation algorithms is mapping the VMs to hosts in an energy-efficient manner. This work advanced the existing work by modeling the algorithms to be independent of workload types and do not need to know the VM application information in advance. However, the algorithms have not been evaluated under realistic testbeds. Based on the VM consolidation approaches in this work, other works like [34][43][71], have done some extension work to improve algorithm performance.

Mastroianni et al. [105] introduced a self-adaptive method for VM consolidation on both CPU and memory. The method aims to reduce the overall costs caused by energy-related issues. The VM consolidation process is determined by a probabilistic function based on the Bernoulli trial. Both the mathematical analysis and realistic testbed results show that the proposed method reduces total energy consumption efficiently.

Zheng et al. [155] jointly considered VM consolidation and traffic consolidation together to minimize the servers and network energy consumption in data centers. The authors not only model the server power model, but also the switch model in the network. Experiments conducted under real environment show that this joint approach outperforms the approaches that only adopt VM consolidation in energy consumption and service delay. Ferdaus et al. [62] proposed a VM consolidation algorithm combining with Ant Colony Optimization, in which a number of artificial ants select feasible solutions and exchange information for their solutions quality to obtain an optimized solution. As the authors consider multiple resource types, the VM consolidation process in this work is modeled as a multi-dimensional vector packing process.

The difference of DVFS and VM consolidation lies in that DVFS achieves energy saving by adjusting frequencies of processors rather than using fewer active servers. The DVFS approach introduces a trade-off between energy consumption and computing performance, where processor lowers the frequency/voltage when it is lightly loaded and utilizes full frequency/voltage when heavily loaded.

Kim et al. [83] modeled real-time service as real-time VM requests. To balance the energy consumption and price, they proposed several DVFS algorithms to reduce energy consumption. Pietri et al. [125] introduced another energy-aware workflow scheduling approach using DVFS and its objective is finding an available frequency to minimize energy consumption while ensuring user deadline. Deng et al. [51] coordinated CPU and memory together to investigate performance constraints, which is the first trial to consider them together when applying DVFS. They aim to find the most energy efficient frequency while ensuring system performance constraints.

To reduce energy consumption, an approach that combines DVFS and VM consolidation together was presented in [140]. The authors proposed several heuristic algorithms for batch-oriented scenarios. A DVFS-based algorithm for consolidating VMs on hosts is introduced to minimize the data center energy consumption while ensuring Service Level Agreement of jobs. The results demonstrate that these two techniques can work together to achieve better energy efficiency.

VM consolidation and DVFS have been proven to be efficient to reduce energy consumption, however, both of them cannot function well when the whole data center is overloaded. Therefore, we introduce a paradigm, called brownout, to handle data center overloads and reduce energy consumption. Originally, the brownout is applied to prevent blackouts through voltage drops in case of emergency. In Cloud scenario, it is first borrowed in [85] to design more robust applications under the overloaded or unpredicted situation. Tomas et al. [142] introduced a combined brownout-overbooking approach to improve resource utilization while ensuring response time. In our previous work, we applied brownout to save energy consumption in data centers. In Chapter 3, we presented the brownout enabled system model and proposed several heuristic policies to find the microservices or application components that should be deactivated for energy saving purpose. We also introduced that there was a trade-off between energy consumption and discount in our model. In Chapter 4, we extended our previous work and adopted approximate Markov Decision Process to improve the aforementioned trade-off. Both in Chapters 3 and 4, the experiments are conducted under simulation environments. Different from them, in this chapter, we implement a prototype system based on real infrastructure.

Some other works related to energy-aware resource scheduling in Clouds are also proposed in the literature. Gai et al. [64] presented a cost-aware heterogeneous cloud memory model to provision memory services and considered energy performance. In [63], the authors introduced a novel approach that aimed to reduce the total energy cost of heterogeneous embedded systems in mobile Clouds. A dynamic energy-aware model to reduce the additional power consumption of wireless communications in the dynamic network environment was introduce in [65]. Sampaio et al. [130] introduced a interference-aware and power-aware approach called PIASA to address the problem that when resource allocation is applied to different applications and workloads, the QoS of applications on VMs in data centers is difficult to be guaranteed. Liao et al. [92] proposed an energy efficient algorithm for distributed storage system. The algorithm aims to save energy and negotiates the QoS constraints as well. The motivation of this work is switching the idle hosts to sleep mode without degrading the QoS requirements of data files and tasks. Different from our work, these articles are not focused on data center energy consumption.

In this chapter, our objective is reducing data center energy consumption while ensuring Quality of Service (QoS). Some related work considering power and QoS have also been conducted. Khanouche et al. [82] proposed an energy-aware and QoS-aware service selection algorithm, which is designed to solve a multi-objective optimization problem. But it is applied to the Internet of Things rather than data centers. Wang et al. [148] used an improved particle swarm optimization algorithm to develop an optimal VM placement approach involving a tradeoff between energy consumption and global

QoS guarantee for data-intensive services in national cloud data centers.

Different from the energy efficient approaches based on VMs, our implementation is based on containers. Compared with VMs, containerization provides cloud application management based on lightweight virtualization. Currently, most work related to containers are focused on the orchestration of containers construction and deployment [119]. A detailed comparison of related work is shown in Table 5.1.

To the best of our knowledge, our work is the first prototype system to reduce energy consumption with brownout based on containers, which also considers the trade-offs between energy consumption and QoS. Our prototype system provides practice and experience for finding complementary option apart from VM consolidation and DVFS.

5.3 Motivations: Scenarios and Challenges

To study service providers' requirement and concerns for managing services based on containers, we give a motivation example of a real-world case study with brownout technology.

A good example of the container-based system is the web-based service. An online shopping system implemented with containers are presented in [2], which contains multiple microservices, including user, user database, payment, shipping, front-end, orders, carts, catalog, carts database and etc. As it is implemented with microservices, each microservice can be activated or deactivated independently. When requests are bursting, the online shopping system may be overloaded, and it cannot satisfy QoS requirements. To handle the overloads and reduce energy consumption, the brownout approach can be applied to temporarily disable some microservices, such as the recommendation engine, to save resource and power. By deactivating the recommendation engine, the system is able to serve more requests with the essential requirement and satisfy QoS. When the system is not overloaded anymore, the disabled microservices are activated again. Considering the overloaded situation, we assume that the service provider of this online shopping system is interested to improve QoS and save energy costs. In addition, the service provider can prefer to apply brownout to manage microservices in their systems. For such deployment, the service provider faces several challenges as below:

1. How to predict the tendency of future workload. It is common for cloud data centers meeting unexpected loads, which can lead to the overloaded situation and performance degradation. Estimating the workloads precisely enables the service providers to select the proper resource management policy.

2. When to disable microservices. Microservices can be dynamically deactivated or activated according to system conditions. A crucial decision should be made in both situations to determine the best time to deactivate containers to relieve overloads and reduce energy consumption while ensuring predefined QoS constraints.

3. Which microservice to disable. Firstly, mandatory and optional microservices are required to be identified. The mandatory microservices, like the database, must be kept running all the time. While the optional microservices are allowed to be deactivated temporarily, such as the recommendation engine in the online shopping system. Secondly, once brownout is triggered, it may require selecting one or more microservices to deactivate. The challenge lies in determining the proper combinations of deactivated microservices to achieve the best beneficial results.

4. When to turn the hosts on or into low-power mode. To reduce energy consumption, it is required to combine brownout and dynamically turning hosts into low power states, which saves the energy of idle hosts. To ensure QoS, it is also essential to determine efficiently when the host states should be switched, because hosts are required to be turned on quickly when requests are increasing.

5. How to design scheduling policy based on brownout. In brownout-compliant microservices, there is a control knob called dimmer that represents a certain probability and shows how often the optional components are executed. It is required to design the dimmer value to be efficiently computed, which supports the brownout to be triggered quickly. The designed policy is also needed to be available for different preferences, like investigating the trade-offs between energy consumption and QoS.

To address aforementioned issues and enable system deployment based on containers and brownout, we introduce our approach: iBrownout.

5.4 iBrownout Architecture

The architecture of iBrownout is demonstrated in Figure 5.1 and its main components are explained below:

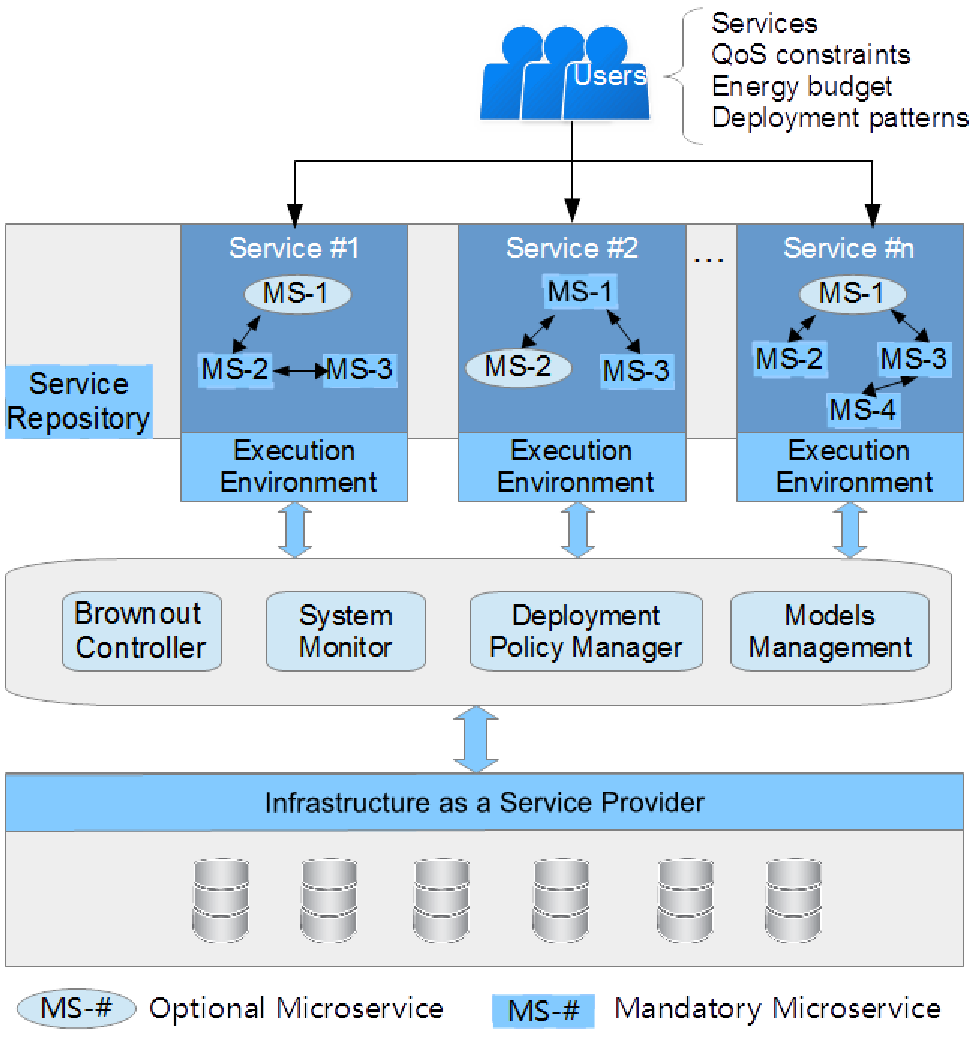


Figure 5.1: iBrownout Architecture

1) **Users:** All services provided by the system are available for users to submit their requests to cloud data centers. The user component contains user' information and requested services. In addition, the system administrator is also included in this component, in which it captures administrators' configurations such as predefined QoS constraints (including maximum response time, error rates and etc.), energy budget and service deployment patterns (in Docker, it is represented as a compose file [10]).

2) **Cloud Service Repository:** The services provided by the service provider are man-

aged by Cloud Service Repository component, which contains the service information, including service's name and image. Each service may be constructed by several microservices, for example, in the online shopping system, the carts service manages items in user's cart, which contains cart microservice showing items in carts and cart database microservice storing items information. To manage microservices with brownout, the microservices are identified as mandatory or optional.

a. Mandatory microservices: The mandatory microservice keeps running all the time when it is launched, such as database-related microservices.

b. Optional microservices: The optional microservices are allowed to be activated or deactivated according to system status. Optional microservices have parameters like CPU utilization $u(MS_c)$, which indicates the amount of CPU usage when it is running and the reduced amount of CPU usage if it is deactivated.

3) Execution Environment: It represents the running environment for containerized applications. The dominant environments are Docker, Kubernetes and Mesos. In our prototype system, we adopt Docker to provide the execution environment for containers/microservices.

4) Brownout Controller: The operations of optional microservices are controlled by Brownout Controller, which determines operations based on system overloaded status. The Brownout Controller takes advantage of scheduling policies that are introduced in Section 5.6 (Scheduling Policies) to offer an elegant solution for operating optional microservices. It is also responsible for monitoring the health of all services. To adapt to our architecture, our dimmer in Brownout controller is different from the one in [85] that requires a dimmer per application. Our dimmer is only applied to the optional microservices. Moreover, rather than based on response time, our dimmer is computed according to the severity of overloaded hosts (the number of overloaded hosts).

5) System Monitor: This component provides health monitoring of nodes and collects hosts resource usage information. Third party monitoring toolkit can be used to provide a view of host status. For instance, the APIs provided by Grid'5000 [67] (a real cluster infrastructure in France) give users real-time reports on infrastructure metric, including host healthy status, utilization and energy consumption.

6) Scheduling Policy Manager: This component provides a set of scheduling policies for Brownout Controller to schedule containers/microservices. Because there exist energy consumption budget and QoS constraints, we have to design and implement policies targeting for different preferences. For example, when the service provider cares more about QoS, a scheduling policy that focuses on optimizing QoS should be applied.

7) Models Management: It provides energy consumption and QoS models for the system. The power consumption model should be modeled to be relevant to microservice/container utilization, and the QoS model identifies the constraints of QoS. such as response time and error rates.

8) Cloud Infrastructure: In infrastructure as a service model, Cloud providers offer bare metal to support service requests, which host multiple containers/microservices. We take advantage of Grid'5000 clusters as our infrastructure.

In order to realize the proposed architecture, several techniques are utilized.

Java: iBrownout is built by using Java and it benefits from Java's feature to run on any platform with Java Virtual Machine. Components including Brownout Controller, System Monitor, Deployment Policy Manager, and Models Management are all implemented with Java. These components calls Docker APIs to collect containers information, such as utilization of containers.

Docker [11]: iBrownout takes advantage of Docker Swarm cluster to manage the containers/microservices, including microservices deployment, stop, start, update and etc. Docker compose file is used to define features of containers, such as whether containers are optional, which containers are deployed, how many containers are provided, how much resources are allocated to containers, deployment constraints of containers and dependencies between different containers.

Ansible [8]: It is a toolkit to automate applications provisioning, configuration management and application deployment. iBrownout utilizes it to send management operations among nodes.

5.5 Modelling and Problem Statement

In this section, we will introduce the models in our system and state the problem we aim to optimize. Table 5.2 presents the symbols and their meanings used in this chapter. For example, we use h_i to denote host i and $P_i(t)$ to represent the power of h_i at time interval t.

5.5.1 Power Consumption

We adopt the servers power model derived from [155]. The power of server i is $P_i(t)$ that is dominated by the CPU utilization:

$$P_i(t) = \begin{cases} P_i^{idle} + u_i \times P_i^{dynamic} & , N_i > 0 \\ 0 & , N_i = 0 \end{cases} \tag{5.1}$$

$P_i(t)$ is composed of idle power and dynamic power. The idle power is regarded as constant and the dynamic power is linear to the server utilization u_i [155]. If no container or microservice is hosted on a server, the server is turned off to save power. The server CPU utilization equals to total CPU utilization of all the containers/microservices deployed to the server, which is represented as:

$$u_i = \sum_{j=1}^{N_i} u(MS_{i,j}(t)) \tag{5.2}$$

where $MS_{i,j}$ refers to the jth microservice on server i, N_i represents the number of microservices deployed to server i. And $u(MS_{i,j}(t))$ refers to the CPU utilization of the container/microservice at time interval t.

Then the total energy consumption during time interval t, with M servers is:

$$E(t) = \sum_{i=1}^{M} \int_{t-1}^{t} P_i(t)dt \tag{5.3}$$

5.5.2 Quality of Service

To model the QoS requirement in our system, we adopt several QoS metrics as below:

Overloaded Time Ratio: Based on host loads, we define two states for hosts: overloaded and non-overloaded. Overloads will lead hosts to experience performance degradation. We regard host as overloaded when host utilization is above the predefined uti-

Table 5.2: Symbols and their meanings

Symbols	**Meanings**
h_i	Server (host) i
t	Time interval t
$P_i(t)$	Power of h_i at time t
P_i^{idle}	Power when h_i is idle
$P_i^{dynamic}$	Power when h_i is fully loaded
P_i^{max}	Maximum power of h_i
hl	Server list in data center
M	Size of server list hl
N_i	Number of microservices assigned to h_i
u_i	Utilization of host h_i
$MS_{i,j}$	Microservice j on h_i
$u(MS_{i,j})$	Utilization of microservice j on h_i
$E(t)$	Energy consumption at time interval t
u_t	Overloaded threshold of host
$OTR(u_t)$	Overloaded time ratio according to u_t
k	Maximum percentile value of response time
t_v	Time threshold of SLA violation
$SLAVR(t_v)$	SLA violation ratio according to violation time threshold t_v
Num_v	The number of requests that violate SLA
Num_a	The total number of requests from clients
C	The maximum number of containers on hosts
α	The maximum allowed overloaded time ratio
β	The maximum allowed average response time
ϕ	The maximum allowed 95th percentile of response time
γ	The maximum allowed SLA violation ratio
M_a	The number of current active hosts
M_a'	The updated number of active hosts for Auto-scaling policy
n_o	Overloaded threshold of request number based on profiling data
n_r	Request rate
$ocl_{i,t}$	The optional container/microservice list on h_i at time interval t
$\mathbb{P}(ocl_{i,t})$	The power set of $ocl_{i,t}$
$dcl_{i,t}$	The deactivated container/microservice list on h_i at time interval t
$HUM()$	Host utilization model to compute host power based on host utilization
HP	The expected host power calculated by host utilization model
TP	The overloaded power threshold
u_i^r	The expected utilization reduction
$u(dcl_{i,t})$	The utilization of deactivated container/microservice list
n_t	The number of overloaded hosts at time interval t
θ_t	The dimmer value
$COH()$	Compute overloaded hosts
$HPM()$	Host power model to compute host utilization based on host power
P_i^r	Expected power reduction of h_i
MS_c	Container/microservice c
S_t	The set of deactivated containers/microservice connection tags
$Ct(MS_c)$	Connection tag of MS_c
X	Random variable to generate sublist of $ocl_{i,t}$

lization threshold. To evaluate this QoS metric to be independent of workloads, we adopt the metric introduced in [32], which is denoted as Overloaded Time Ratio (OTR):

$$OTR(u_t) = \frac{t_o(u_t)}{t_a} \tag{5.4}$$

where u_t is the overloaded CPU utilization threshold; t_o is the time period that host is identified as overloaded, which is relevant to u_t; and t_a is the total time periods of the hosts. As a QoS constraint, this metric is configured as the maximum allowed value of OTR. For instance, if the system SLA is defined as 10%, the time period of overloaded states for all the hosts is less then 10%. The SLA constraint can be formulated as:

$$\frac{1}{M}\sum_{i=1}^{n=M} OTR_n(u_t) \leq 0.1 \tag{5.5}$$

where M is the total number of hosts in the data center. As introduced in the later sections, our brownout-based approach checks the host status at each time period and triggers the brownout to deactivate containers when there are overloaded hosts. Therefore, this metric also represents the ratio that brownout is triggered.

Response time: This metric measures the time that from sending requests to receiving requests. We also evaluate the response time with the maximum of kth percentile response time of all requests, where k could be 90, 95, 99 and etc. For example, if the maximum of 95th percentile response time equals to 1 second, it means that 95% of all requests get the response within 1 second.

SLA Violation Ratio: It represents how many requests are failed due to overload. If clients send Num_a requests to the system, and Num_{err} of them are returned with errors, then the error rate is represented as:

$$SLAVR = \frac{Num_{err}}{Num_a} \tag{5.6}$$

5.5.3 Optimization Objective

As discussed in the previous section, it is necessary to minimize the total energy consumption, while ensuring QoS by avoiding overloads, decreasing response time and reducing error rates. Therefore, our problem can be formulated as an optimization problem (5.7)-(5.10):

$$\begin{aligned} &\min \sum_{t=1}^{T} E(t) \\ s.t. \frac{1}{M} &\sum_{n=1}^{n=M} OTR_n(u_t) \ \leq \alpha, \forall t \\ &R^t_{avg} \leq \beta,\ R^t_{95th} \leq \phi,\ \forall t \\ &SLAVR \leq \gamma \end{aligned} \tag{5.7}$$

where $\sum_{t=1}^{T} E(t)$ is the total energy consumption of data center, α is the maximum allowed average response time of overloaded states; R^t_{avg} is the average response time and β is the allowed average response time; R^t_{95th} is the maximum of 95th percentile response time and ϕ is the allowed the 95th percentile response time, and γ is the allowed SLA violation ratio.

5.6 Scheduling Policy

In this section, we will introduce our brownout-based scheduling policies. Prior to brownout approach, we require an auto-scaling algorithm to dynamically add or remove hosts to utilize host resource more efficiently.

5.6.1 Auto-scaling Policy

We adopt the auto-scaling algorithm in [143], which is a predefined threshold-based approach. With profiling experiments, we configure the requests overloaded threshold above which the host cannot respond to requests within an acceptable time limit. As shown in Algorithm 5.1, in the initialization stage, the master node that runs auto-scaling algorithm firstly gets the number of current active hosts (line 1), sets the overloaded threshold of request number according to profiling data (line 2) and fetches the request rate at current time window according to previous time windows (line 3). The advantage

Algorithm 5.1: Auto-scaling Policy

```
Input  : host list $hl$ with size $M$, number of active hosts $M_a$, number of requests when host is
         overloaded $n_o$, recent request rate in the recent time $n_r$.
Output: number of active hosts $M_{a'}$
1  $M_a \leftarrow$ number of current active hosts
2  $n_o \leftarrow$ overloaded threshold of request number according to profiling data
3  $n_r \leftarrow$ number of request rate at current time window according to previous time windows
4  $M_{a'} \leftarrow \lceil n_r \div n_o \rceil$
5  $M' \leftarrow M_{a'} - M_a$
6  if $M' > 0$ then
7      Add $M' hosts$
8  else if $M' < 0$ then
9      Remove $|M'| hosts$
10 else
11     no scaling
12 end
13 update number of active hosts with $M_{a'}$
```

of sliding time window is to give more weights to the values of recent time windows, and more details will be given in Section 5.7. Line 4 shows the method to compute the current required hosts $M_{a'}$, which is the ratio of current request rate and the overload threshold. If the required number of hosts is more than current active hosts, more hosts will be added to provide services, otherwise, if current active hosts are more than required, then the excess machine can be set as low-power mode to save energy consumption (lines 6-12). Finally, the master node will update the number of active hosts.

5.6.2 Initial Deployment

In the initial deployment stage, containers are deployed based on Docker compose file, which identifies all the required information of services and the configurations of initial deployment. A simple example is shown in Figure 5.2. Lines 2-14 show the information of recommendation engine service, which is built on the Ubuntu image and attached with a data volume. The recommendation engine is set as optional microservice, which can be deactivated and has two replicates. Moreover, this service will only be deployed on Docker worker node as deployment constraint. Lines 16-21 demonstrate the information of user database service, which is not optional and restricts to be deployed to Docker master node.

```
services:
  recommendation-engine:
    image: ubuntu
    tty: true
    volumes:
    - DataVolume:/DataVolume
    labels:
      brownout.feature: "optional"
    deploy:
      replicas: 2
      restart_policy:
        condition: none
      placement:
        constraints: [node.role == worker]

  user-db:
    image: weaveworksdemos/user-db
    hostname: user-db
    deploy:
      placement:
        constraints: [node.role == manager]
```

Figure 5.2: Simple example of Docker compose file

5.6.3 Optimization Deployment with Scheduling Policies

We have proposed three brownout-based policies as follows:

Lowest Utilization Container First (LUCF)

The Lowest Utilization Container First policy selects a set of containers with the lowest utilization that reduces the utilization to be less than the overloaded threshold of a host is overloaded. Let $ocl_{i,t}$ be the optional container list on host h_i at time interval t. Let $\mathbb{P}(ocl_{i,t})$ to be the power set of $ocl_{i,t}$, the LUCF finds the deactivated container list $dcl_{i,t}$, which is included in $\mathbb{P}(ocl_{i,t})$. The deactivated container list minimizes the value difference between the expected utilization reduction u_i^r and its utilization $u(dcl_{i,t})$ The deactivated container list is defined in Equation 5.8.

$$dcl_{i,t} = \begin{cases} \{HP \leq TP, u_i^r - u(dcl_{i,t}) \rightarrow \min\}, & if\ P_i(t) \geq TP \\ \emptyset, & if\ P_i(t) < TP \end{cases} \tag{5.8}$$

Algorithm 5.2: Lowest Utilization Container First Policy (LUCF)

```
Input  : host list $hl$ with size $M$, microservice information, overloaded power threshold $TP$,
         dimmer value $\theta_t$ at time $t$, scheduling interval $T$, deactivated component list $dcl_{i,t}$ on host
         $h_i$, power model of host $HPM$, the optional component list $ocl_{i,t}$, which is sorted based on
         utilization $u(MS_c)$ in ascending order
Output: total energy consumption, number of shutting down hosts
1  initialize parameters with inputs, like $TP$
2  for $t \leftarrow 0$ to $T$ do
3      $n_t \leftarrow COH(hl)$
4      if $n_t > 0$ then
5          $\theta_t \leftarrow = \sqrt{\frac{n_t}{M}}$
6          forall $h_i$ in $hl$ (i.e. $i = 1, 2, \ldots, M$) do
7              if ($P_i(t) > TP$) then
8                  $P_i^r \leftarrow \theta_t \times P_i(t)$
9                  $u_i^r \leftarrow HPM(h_i, P_i^r)$
10                 $dcl_{i,t} \leftarrow$ NULL
11                 $S_t \leftarrow$ NULL
12                 if $u(MS_1) \geq u_i^r$ then
13                     $dcl_{i,t} \leftarrow dcl_{i,t} + MS_1$
14                     $S_t \leftarrow S_t + Ct(MS_1)$
15                 end
16                 for $MS_c$ in $ocl_{i,t}$ do
17                     if $(u(MS_c) \leq u_i^r)$ & $(u(dcl_{i,t}) \leq u_i^r)$ then
18                         $dcl_{i,t} \leftarrow dcl_{i,t} + MS_c$
19                         $S_t \leftarrow S_t + Ct(MS_c)$
20                         min $\leftarrow (u_i^r - u(dcl_{i,t}))$
21                     end
22                 end
23                 forall $MS_c$ in $ocl_{i,t}$ do
24                     if $Ct(MS_c)$ in $S_t$ then
25                         $dcl_{i,t} \leftarrow dcl_{i,t} + MS_c$
26                     end
27                 end
28             end
29             deactivate components in $dcl_{i,t}$
30         end
31     else
32         activate deactivated components
33     end
34 end
```

Table 5.3: Power consumption of selected node at different utilization levels in Watts

Utilization	Sleep	**0%**	**10%**	**20%**	**30%**	**40%**
Power (Watts)	10	201	206	211	213	216
Utilization	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
Power (Watts)	221	223	225	231	233	237

where HP is the expected host power calculated by host utilization model $HUM(h_i, u_i - u(dcl_{i,t}))$ that fetches the host power based on host utilization $u_i - u(dcl_{i,t})$; TP is the overloaded power threshold of h_i.

The pseudocode of LUCF is shown in Algorithm 5.2, which mainly consists of 8 steps as discussed below. Before entering the approach procedures, service provider firstly needs to initialize input parameters for the algorithm, such as overloaded power threshold (lines 1-2). The power threshold TP is a value for checking whether a host is overloaded.

1) In each time interval t, checking all the hosts status and counting the number of overloaded hosts as n_t (line 3).

2) Adjusting the dimmer value θ_t as $\sqrt{\frac{n_t}{M}}$ based on the number of overloaded hosts n_t and host size M (line 5). As introduced in related work, the dimmer value θ_t is applied to compute the adjustment degree of power consumption at time t. The dimmer value θ_t is 1.0 if all the hosts are overloaded at time t and it means that brownout controls containers/microservice on all the hosts. The dimmer value is 0.0 if no host is overloaded and brownout will not be triggered at time t. The adjustment of dimmer presents that the dimmer value is relevant to the number of overloaded hosts.

3) Calculating the expected utilization reduction on the overloaded hosts (lines 7-9). Based on the dimmer value and host power model, LUCF calculates the expected host power reduction P_i^r (line 8) and the expected utilization reduction u_i^r (line 9) respectively. In our host power model, the host power consumption is mainly relevant to it CPU utilization. As shown in Table 5.3, we list power consumption at different CPU utilization levels of one host in Grid'5000 (Sagittare cluster in Lyon). In this power model, for example, the host with 100% utilization is 237 Watts and 80% utilization is 231 Watts, if the power is required to be reduced from 237 to 231 Watts, the expected utilization reduction is $100\% - 80\% = 20\%$.

4) Resetting the deactivated container list $dcl_{i,t}$ and the set of deactivated container connection tags S_t as empty (lines 10-11). This list and the set will be ready to collect deactivated containers and their connection tags.

5) Finding the containers to be deactivated (lines 16-27). The LUCF sorts the optional container list $ocl_{i,t}$ based on container utilization parameter in ascending order , therefore, the container with the lowest utilization is put in the head of the list. Since we consider connected containers, each container has a connection tag $Ct(MS_c)$ that shows how it is connected with other containers. If the first container utilization parameter value is above u_i^r, Algorithm 5.2 adds this container into the deactivated container list $dcl_{i,t}$ and inserts its connection tag $Ct(MS_1)$ into S_t (lines 12-13). After that, Algorithm 5.2 finds other connected containers and adds them into deactivated container list (line 14). If the first container utilization does not satisfy the expected utilization reduction, Algorithm 5.2 finds the containers sublist in the optional container list to deactivate more containers (lines 16-22). The utilization of this sublist is closest to the expected utilization reduction among all the sublists.

Algorithm 5.2 also puts all the containers in the sublist into the deactivated containers list and puts their connection parameters into the S_t. For connected containers, the sorting process is modified as treating the connected containers together for sorting, which lowers the priority of deactivating the connected containers, and avoids deactivating too many containers due to connections.

6) Finding other connected container and putting them into the deactivated container list (lines 23-27).

7) Deactivating the containers in the deactivated container list (line 29).

8) In Algorithm 5.2, if no host is above the power threshold, the algorithm activates the deactivated containers (line 32).

It is noticed that when the whole data center is overloaded, auto-scaling cannot add more hosts because of the limited resource. LUCF takes effects when Auto-scaling cannot function well, to be more specific, LUCF can be embedded into line 7 in Algorithm 5.1 to handle with overloads and reduce energy consumption.

Algorithm Complexity: the complexity of LUCF at each time interval is calculated as below: the complexity of finding the deactivated containers is $O(C * M)$, where C is the maximum number of containers on hosts and M is the number of hosts. The complexity of finding the connected components is also $O(C * M)$. Therefore, the complexity at each time interval of LUCF is the sum of these parts, which is $O(2 * C * M)$. To be noted, line 3 relies on the network connection, if C and M are small, the network delay $O(T_d)$ can be a dominant part of algorithm execution time. Please see the results in Section 5.7.4.

Minimum Number of Components First Policy (MNCF)

The Minimum Number of Containers First (MNCF) policy selects the minimum number of containers while reducing the energy consumption in order to deactivate fewer services, as formalized in Equation 5.9. We do not provide the pseudocode of MNCF here because it is quite similar to the LUCF algorithm introduced earlier.

$$dcl_{i,t} = \begin{cases} \{HP \leq TP, |u(dcl_{i,t})| \rightarrow \min\}, & if\ P_i(t) \geq TP \\ \emptyset, & if\ P_i(t) < TP \end{cases} \tag{5.9}$$

Random Selection Container Policy (RSC)

The Random Selection Container policy (RSC) policy takes advantage of a random selection of a number of optional containers to reduce energy consumption. Based on a uniformly distributed discrete random variable (X), which selects randomly a subset of $dcl_{i,t}$, RSC is presented in Equation 5.10.

$$dcl_{i,t} = \begin{cases} \{HP \leq TP, X = U(0, |ocl_{i,t}| - 1)\}, & if\ P_i(t) \geq TP \\ \emptyset, & if\ P_i(t) < TP \end{cases} \tag{5.10}$$

5.7 Performance Evaluation

We are evaluating our techniques experimentally on INRIA Grid'5000 testbed for Wikipedia web workload. We also compare the performance with related policies introduced in [33], [142] and [143].

5.7.1 Workload

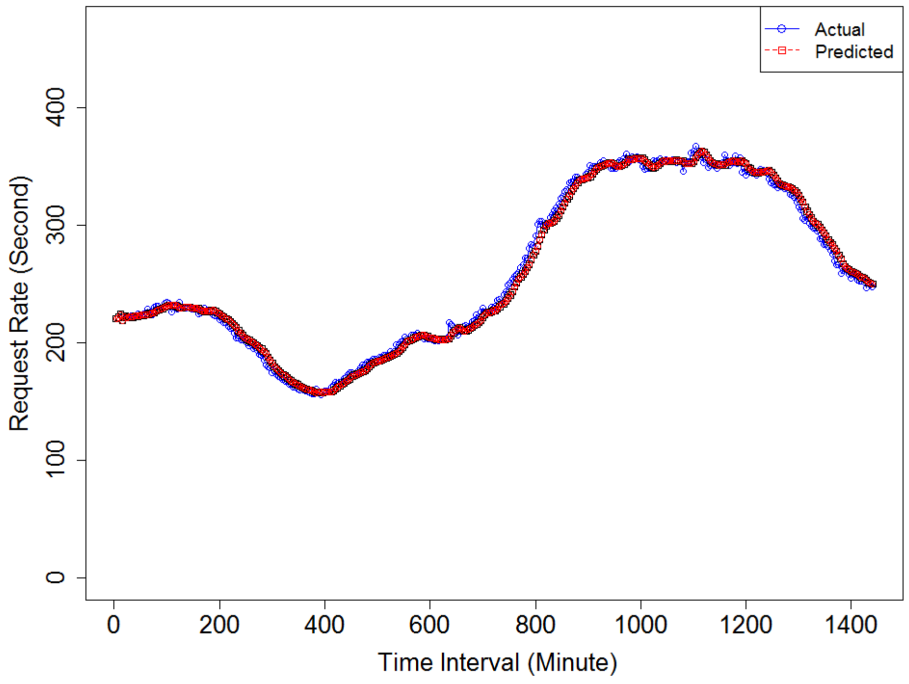


Figure 5.3: Predicted and actual requests rate

We use real trace from Wikipedia requests on 2007 October 17 to replay the workload of Wikipedia users. To scale the workload set to fit with our experiments, we use 5% of the original user requests size. JMeter [9] is a toolkit designed for load testing and performance measurement, we use it to generate the requests by replaying the Wikipedia trace. n_r is the predicted request rate, which is calculated based on a sliding window [32]. Let L_w to be the window size, and $n_r(t)$ to be the request rate at t, we estimate n_r as:

$$n_r(L_w) = \frac{1}{L_w} \sum_{t=0}^{L_w - 1} n_r(t) \tag{5.11}$$

In our experiments, we set the sliding window size as 5. Figure 5.3 shows the requests rate per second during the day, and the predicted rates and the actual rates are quite close.

5.7.2 Testbed

We use Grid'5000 [67], a French experimental grid platform, as our testbed. We adopt the cluster equipped with power measurement APIs at Lyon site, which is located at the southeast French. The architecture of prototype system deployed on the Grid'5000 clusters is presented in Figure 5.4, which shows that all the nodes are deployed with Docker swarm and categorized according to different roles as below:

- Master node: this node is initialized as the master node and running some services that can only be deployed on the master node, such as the brownout controller containing scheduling policies, as well as the Java Runtime and Ansible toolkit.
- Worker node: these nodes are workers that running services apart from the services on master node and database services. We have multiple worker nodes in our system.
- Worker node (node only for the database): the database services are deployed on a specific worker node, which only hosts database-related services.

We also have another node, namely request node, that contains workload trace and installed with JMeter to send requests to our cluster. This node can be located at any place to simulate users' behavior. In our experiments, to reduce the impacts of uncontrolled network traffic out of Lyon cluster, we also locate this node in Lyon cluster.

The hardware information of our selected nodes is as below:

- Machine model: Sun Fire V20z. The maximum power of this model is 237 Watts, and its power of sleep mode is 10 Watts;
- Operating system: Debian Linux;
- CPU: AMD Operon 250 with 2 cores (2.4 GHz);
- Memory: 2 GB

One of the nodes is running as the Docker Swarm master node, and other nodes are running as worker nodes. All required applications, such as Java, Docker, Ansible and JMeter, are installed in advance to minimize the impacts of CPU utilization and network traffics.

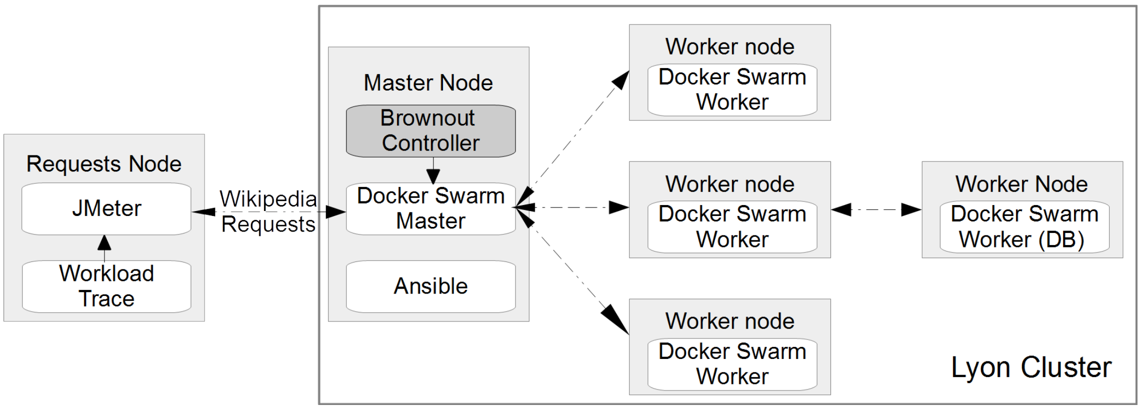


Figure 5.4: Architecture of Prototype System

5.7.3 Results

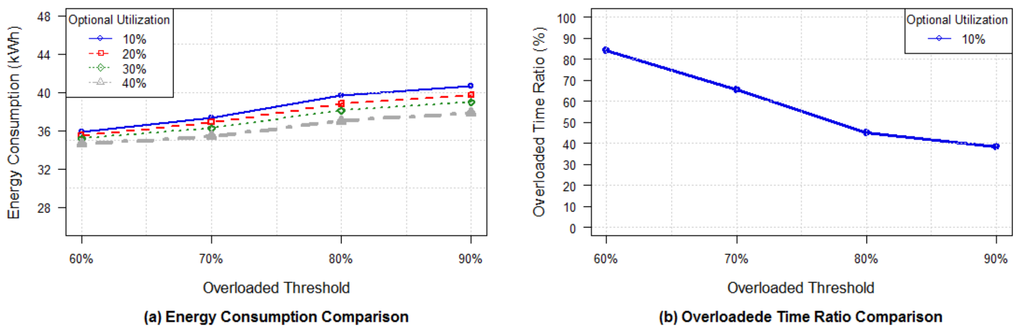


Figure 5.5: Algorithm Performance Comparison

To evaluate the performance of our proposed policies, we use three benchmark policies for comparison.

1). **Non-Power-Aware (NPA)** policy [33]: It applies no power-aware optimization and hosts are keeping on all the time. We give 13 nodes as the resource for NPA.

2). **Brownout-OverBooking (BOB)** policy [142]: It aims to maximize actual utilization while reducing response time and minimally triggering brownout. The brownout operation in BOB is based on response time. When the response time is less than target utilization, the approach gradually increases application utilization. To let BOB experience overloads, only 10 nodes are given to it.

3). **Auto Scaling (Auto-S)** policy [143]: It dynamically scales in and out the number of active hosts as introduced in Algorithm 5.1. To let Auto-S endure overloads, we also give 10 nodes to Auto-S.

For our proposed policies, they have the identical resource as BOB and Auto-S. In the following experiments, we mainly investigate two parameters: overloaded threshold and optional utilization percentage.

Overloaded threshold: It represents the CPU utilization threshold that identifies whether a host is overloaded. We adopt this parameter since [33] have shown that it has an impact on energy consumption. It is varied from 60% to 90% in increments of 10%. We choose this range because of the smaller overloaded threshold, like 50%, means hosts are easier to be identified as overloaded and it will lead to inefficient resource usage.

Optional utilization percentage: It identifies how much CPU resource is given to optional containers, which also means how much CPU utilization can be reduced to save energy consumption. This parameter is investigated because Chapter 3 shows that it influences the power consumption. It is varied from 10% to 40% in increments of 10%. We choose these ranges because Chapter 3 shows large optional utilization percentage, like 50%, comes along much revenue loss and non-negligible experience degradation.

(1) Comparison with different overloaded thresholds

We have conducted several experiments with different values of overloaded threshold and optional utilization percentage for LUCF policy. In Figure 5.5, the results show that when the overloaded threshold is higher, LUCF reduces less energy consumption, and when the system has higher optional utilization percentage, LUCF saves more energy consumption. However, as shown in Figure 5.5b, when the overloaded threshold is smaller, like 60%, the overloaded time ratio is quite high (around 85%), which means hosts are regarded as overloaded in most time periods and brownout will be triggered frequently. As optional utilization percentage does not influence overloaded time ratio, we only show the LUCF with 10% optional utilization here. From the results, we observe a trade-off between energy consumption and overloaded ratio time when the overloaded threshold is varied, and we find out that configuring the overloaded threshold as 70% and 80% achieves better trade-offs, which reduces energy consumption while not triggering brownout too frequently. Therefore, we conduct experiments under 70% and 80% overloaded thresholds to compare our proposed policies in the following section.

(2) Comparison with proposed policies

Figure 5.6 shows the results with varied overloaded thresholds and optional utiliza-

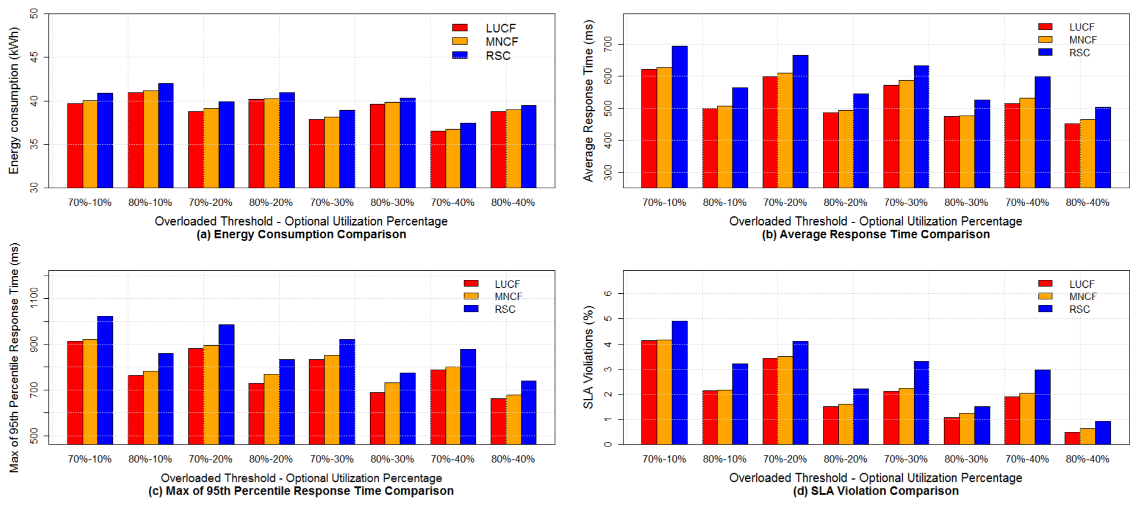


Figure 5.6: Performance Comparison of Proposed Policies

tion percentages for our proposed policies, we compare the energy consumption, average response time, maximum of 95th percentile response time and SLA violations achieved by LUCF, MNCF and RSC. For the energy consumption, under same optional utilization percentage, policies with 70% overloaded threshold save more energy than policies with 80%. For example, when the optional utilization percentage is 10%, LUCF with 70% overloaded threshold has 39.7 kWh and LUCF with 80% overloaded threshold has 40.9 kWh. It is observed that with more optional utilization percentage, all the policies reduce more energy consumption, and both LUCF and MNCF save more energy consumption and RSC. Under 80% overloaded threshold, as the energy consumption of LUCF and MNCF is quite close, we conduct the paired *t*-tests for them, and the p-values are 0.09, 0.15, 0.1 and 0.09 respectively. Therefore, we conclude that energy consumption of LUCF and MNCF has no statistically significant difference when the overloaded threshold is 80%.

For the comparison of average response time and maximum of 95th percentile response time in Figure 5.6b and Figure 5.6c, policies with 70% overloaded threshold experience more average response time and maximum of 95th percentile response time than the ones with 80% overloaded threshold. The average response time of LUCF with 70% overloaded threshold ranges from 515 to 621 ms, while with 80% overloaded threshold, it is from 452 ms to 500 ms. When more optional utilization percentage is configured, the average response time and the maximum of 95th percentile response time is reduced. For instance, with 80% overloaded threshold, the average response time of LUCF is reduced from 500 to 452 ms, and the maximum of 95th percentile response time of MNCF is decreased from 780 to 680 ms. The results show that brownout-based policies are able to improve response time as well as energy saving. Figure 5.6d illustrates the comparison of SLA violations. When the overloaded threshold is 70% and optional utilization percentage is 10% the SLA violation is more than 4%, as the overloaded threshold and the optional utilization percentage increase, the SLA violations are reduced to less than 1%.

To conclude, LUCF and MNCF achieve better performance than RSC, as RSC selects containers randomly rather than deterministic methods. LUCF and MNCF have close energy consumption, but in most cases, LUCF achieves better performance in response time and SLA violations than MNCF. The reason lies in that LUCF has more container deactivation options than MNCF. For different overloaded thresholds comparison, policies with 70% overloaded threshold save more energy but have the more average response time, maximum of 95th percentile response time and SLA violations than policies with 80% overloaded threshold. Configuring overloaded threshold as 80% achieves a better trade-off than 70%, as it reduces energy consumption while not having large average response time. Thus, the following experiments are conducted under 80% overloaded threshold. Additionally, as LUCF has the best performance among our proposed policies, we choose LUCF as the representative of our proposed algorithms to compare with benchmark policies.

Table 5.4: Final experiment results

Policy	Energy (kWh)	Average response time	Max of 95th response time	SLA violation
NPA	69.71 (68.94,70.45)	**188.8** (137.4, 240.2)	**312.2** (178.8, 445.8)	-
BOB	49.83 (49.06, 50.60)	440.1 (426.0, 454.1)	712.4 (696.8, 727.9)	1.240 (1.098, 1.381)
Auto-S	43.95 (43.48, 44.43)	511.0 (502.3, 519.6)	929.5 (840.9, 1018.1)	4.240 (4.098, 4.382)
LUCF-10	40.36 (40.01, 40.71)	482.1 (471.5, 492.7)	775.4 (746.2, 804.6)	2.140 (2.020, 2.259)
LUCF-20	40.17 (39.87, 40.47)	476.0 (462.4, 489.5)	735.7 (712.2, 759.1)	1.516 (1.340, 1.691)
LUCF-30	39.41 (38.93, 39.89)	451.5 (428.1, 475.0)	721.1 (702.3, 739.9)	1.082 (1.005, 1.158)
LUCF-40	**38.60** (38.21, 39.01)	431.1 (415.0, 447.2)	687.8 (661.2, 714.4)	**0.494** (0.439, 0.548)

(3) Final experiment results

Figure 5.7 and Table 5.4 present the mean values of energy consumption, average response time, maximum of 95th percentile response time and SLA violations along with 95% CI for the NPA, BOB, Auto-S and LUCF with different optional utilization percentages. The results demonstrate that NPA has energy consumption 69.71 kWh with 95% CI

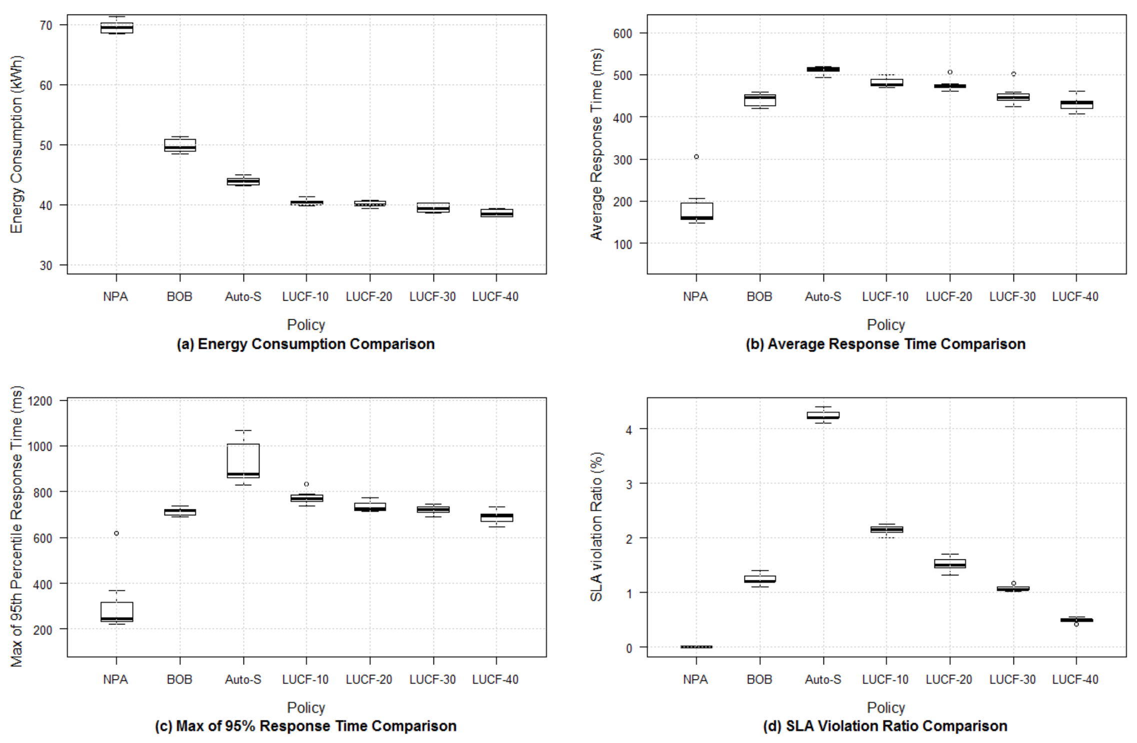


Figure 5.7: Performance comparison of policies

(68.94, 70.45), BOB has 49.83 kWh with 95% CI (49.06, 50.6), and Auto-S reduces it to 43.95 kWh with 95% CI (43.48, 44.43). LUCF saves more energy consumption than Auto-S, to be more specific, LUCF with 10% optional utilization leads to 40.36 kWh with 95% CI (40.01, 40.71) and lowers gradually to 38.6 kWh with 95% CI (38.21, 39.01) when optional utilization is 40%.

In the comparison of average response time and the maximum of 95th percentile response time in Figure 5.7b and Figure 5.7c, as NPA has adequate resources, it has the minimum response time compared with other policies. Its average response time is 188.8 ms with 95% CI (137.4, 240.2) and its maximum of 95th percentile response time is 312.2 ms with 95% CI (178.8, 445.8). As Auto-S experiences overloads, its average response time and the maximum of 95th response time are 511 ms with 95% CI (502.3, 519.6) and 929.5 with 95% CI (840.9, 1018.1) respectively. Taking advantage of brownout, although BOB and LUCF endure overloads, their brownout controllers relieve the overloaded situation. In BOB, its average response time is reduced to 440.1 ms with 95% CI (426.0, 454.1) and its maximum of 95th response time is 712.4 ms with 95% CI (696.8, 727.9). In LUCF with 40% optional utilization percentage, its average response time and the maximum of

95th response time are reduced to 431.1 ms with 95% CI (415, 447.2) and 687.8 ms with 95% CI (661.2, 714.4) respectively. Figure 5.7d presents the SLA violation comparison. NPA does not have SLA violations, BOB has 1.24% with 95% CI (1.098, 1.381), and Auto-S has 4.24% with 95% CI (4.098, 4.382) SLA violations. When more optional utilization is offered, LUCF improves the SLA violations from 2.14% to 0.5% in average values.

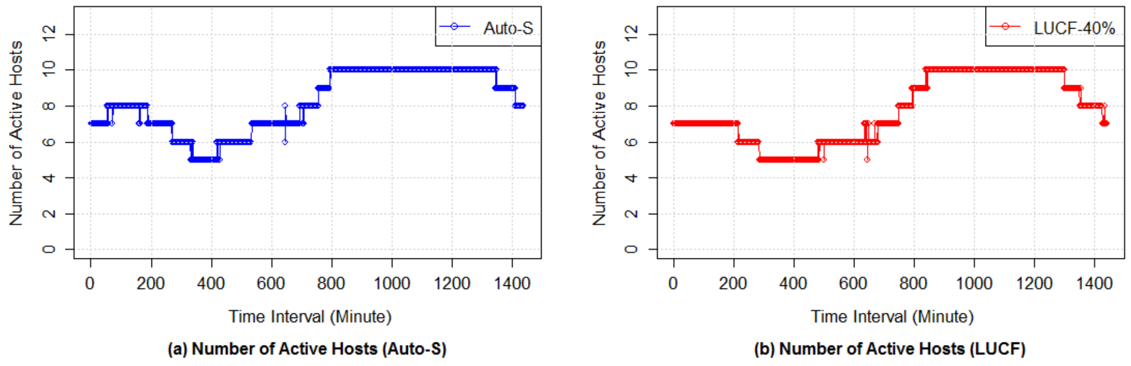


Figure 5.8: Number of active hosts comparison

This is due to the fact that LUCF uses less active hosts as shown in Figure 5.8, which shows the number of active hosts within one day. For instance, at the time intervals from 400-500, 6 hosts are active with Auto-S, while LUCF runs 5 active hosts. For NPA and BOB, hosts are always at active states. From the presented results, we can conclude that the LUCF achieves better energy consumption than NPA, BOB and Auto-S. According to response time and SLA violation comparison, LUCF outperforms Auto-S. Compared with BOB, LUCF has better performance when optional utilization percentage is larger than 30%.

5.7.4 Scalability

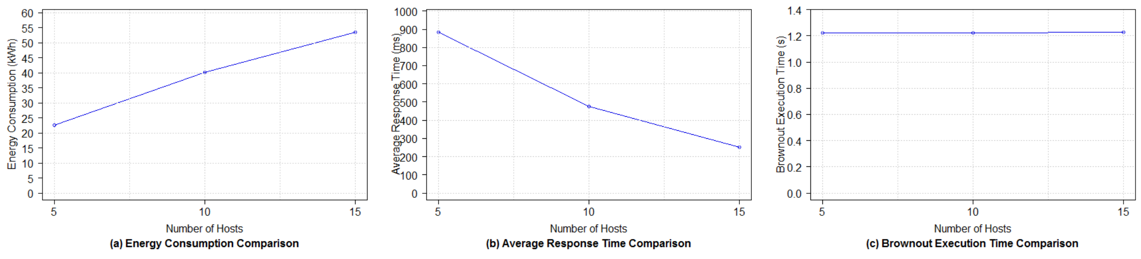


Figure 5.9: Scalability evaluation of iBrownout with LUCF policy

Table 5.5: Scalability experiments results

Number of Hosts	Energy Consumption	Average Response Time	Brownout Execution Time
5 hosts	22.6 kWh	882 ms	1.223421 s
10 hosts	40.2 kWh	476 ms	1.224356 s
15 hosts	53.4 kWh	251 ms	1.224973 s

In this section, we evaluate the scalability of the proposed approach and the efficiency of the algorithm when the number of nodes is increased. As mentioned in previous sections, iBrownout is implemented based on Docker Swarm, thus, its performance depends on the performance of Docker Swarm. Our aim in this chapter is not to discuss the scalability design of Docker Swarm. In [100], the authors conducted scalability testing on Docker Swarm with 1,000 nodes and 30,000 containers, and results show that Docker Swarm has high scalability.

We evaluate the scalability of iBrownout in terms of the number of hosts. The experiment settings are almost as same as in the previous experiments, the overloaded threshold is set as 80% and optional utilization percentage is 30%, while the difference lies in the number of hosts, we conduct experiments with 5, 10 and 15 hosts respectively. Energy consumption and QoS are the main concern of our proposed approach. In this chapter, we only focus on average response time as QoS metric. In addition, to compare algorithm efficiency, we also evaluate the brownout algorithm (LUCF policy) execution time, which represents the time between brownout is triggered and the deactivated components are selected.

Figure 5.9 and Table 5.5 show the impact of the varied number of hosts on energy consumption, average response time and brownout algorithm execution time. As it can be seen, when there are more hosts, the energy consumption is increased and the average response time is reduced, while the brownout execution time is kept as stable. The energy consumption is growing from 22.6 kWh with 5 hosts to 53.4 kWh with 15 hosts, while the average response time is dropping to 251 ms with 15 hosts from 882 ms with 5 hosts. The reason lies in that when more hosts are running, these hosts consume more energy, and the benefit is that the average response time is reduced due to more resources. The brownout execution time remains 1.22 s when the number of hots is varied. As mentioned in Section 5.6.3.1, although the algorithm complexity of LUCF is relevant to the number of hosts, the search operation in LUCF only consumes a small portion of time compared with the network delay to fetch the information of hosts and containers. Therefore, the brownout execution time remains stable when the number of hosts is increased. The results show that iBrownout scales reasonably well when the number of hosts grows. To be noted, the master node in Docker Swarm may be the bottleneck if there are a number of worker nodes but only one master node, thus, more nodes should be promoted as master nodes to ensure the system scalability.

5.8 Summary

In this chapter, we introduced a brownout-based architecture by deactivating optional containers in applications or microservices temporarily to reduce energy consumption. Under this architecture, we introduced an integrated approach for managing energy and brownout in container-based clouds. We also proposed several policies to find the suitable containers to deactivate and evaluate their performance in a prototype system. The experiment results under real test-beds have shown that our proposed policies can achieve better performance in energy consumption, response time and SLA violations than baselines.

After discussing the brownout-based architecture for approaches implemented under real testbed, in the next chapter, we will introduce the architecture, design and implementation of our developed container-based prototype system, which aims to attract more attention in the brownout-related area.

Chapter 6
Container-based Software System for Energy-Efficient Cloud Computing

Container-based approach provides a mechanism to manage microservices in a more fine-grained manner and improve the resource usage of computing systems. Brownout approach can overcome the limitation of VM consolidation and Dynamic Voltage Frequency Scaling when the holistic system is overloaded via dynamically deactivating or activating optional microservices or containers. In this chapter, we propose a brownout-based software system for container-based clouds to handle overloads and reduce power consumption. We also introduce its design and implementation based on Docker Swarm containers. The proposed system is integrated with existing Docker Swarm without the modification of their configurations. In addition, to show the availability to reduce energy, we implement several policies to manage containers and conduct experiments on French Grid'5000 cloud infrastructure. The results show that our policies can save about 40% and 10% energy than two existing baselines while ensuring quality of services.

6.1 Introduction

TOOLKITS and software systems are necessary to foster innovation and development for experiments with microservices and brownout approaches in cloud computing systems. Docker [11] is developed to manage microservices and it enables to develop different resource management policies for microservices. However, the brownout mechanism is not provided in Docker. To overcome this limitation, we develop a container-based software prototype that enables brownout approach.

This chapter is derived from:
• **Minxian Xu** and Rajkumar Buyya, "BrownoutCon: A Software System based on Brownout and Containers for Energy Efficient Clouds," *Journal of Systems and Software (JSS)*, 2019 (under review).

In this chapter, we propose and develop a software system, called BrownoutCon[2], which is inspired by brownout-based approach to deliver energy-efficient resource scheduling. The implementation of BrownoutCon is based on Docker Swarm [11] that provides the management of container cluster. The software system is designed and implemented as an add-on for Docker Swarm, which has no necessity to modify the configurations of Docker Swarm. The system also applies the public APIs of Grid'5000 [67], which is a real testbed that provides power measurement for hosts. The aims of BrownoutCon are twofold: 1) providing an open-source software system based on brownout and Docker Swarm to manage containers; 2) offering an extensible software system for conducting research on reducing energy consumption and handling overloads in cloud data centers.

The BrownoutCon is designed and implemented by following the brownout enabled system model in our previous chapters. Mandatory containers and optional containers are introduced in the system model, which are identified according to whether the containers can be temporarily deactivated or not. The brownout controller is the key part of the system model to manage brownout, which also provides the scheduling policies for containers. The problem of designing the brownout controller splits into several subproblems:

1. Predicting the future workloads, so that the system can avoid overloads to foster the system robustness.

2. Determining whether a host is overloaded or not, so that the brownout controller will be triggered to relieve the overloads.

3. Deciding when to disable the containers, so that the system can relieve overloads and reduce energy consumption while ensuring QoS constraints.

4. Selecting the containers to be disabled, so that a better trade-off can be achieved between the reduced energy and QoS constraints.

5. Deciding when to turn the hosts on or into the low-power mode, so that the idle hosts can be switched into low-power mode to save power consumption.

Compared with VM consolidation approaches, the software system based on brownout and containers has two advantages: 1) a container can be stopped or restarted in sec-

[2]The source codes are available at Github: https://github.com/xmxyt900/BrownoutPrototype

onds, while VM migration may take minutes. Thus, scheduling with containers is more light-weight and flexible than VMs. 2) the brownout-based approach provides another optional energy-efficient approach apart from VM consolidation and DVFS, which is also available to be combined with VM consolidation to achieve better energy efficiency, especially for the situation when the whole data center is overloaded.

To evaluate the proposed system in practice, we conduct our experiments on Grid'5000 [67] real testbed. We also evaluate the performance of proposed system with real traces derived from Wikipedia[3] workloads.

The main **contributions** of this chapter are as follows:

- Proposed an effective system model that enables brownout approach to manage the containers and resources in a fine-grained manner;
- Designed and developed a software system based on Docker Swarm to provide energy-efficient approaches for cloud data centers;
- Experimental evaluations of our proposed software system on French Grid'5000 infrastructure for service providers to deploy microservices in an energy-efficient manner while ensuring QoS constraints.

The rest of this chapter is organized as: Section 6.2 discusses the related work, followed by the system design and implementation in Section 6.3. Brownout-based policies implemented in BrownoutCon are presented in Section 6.4. In Section 6.5, we introduce our experiments setup and evaluate the performance of implemented policies under Grid'5000 testbed. Conclusions along with future work are given in Section 6.6.

6.2 Related Work

It is estimated that U.S. data centers will consume 140 billion kWh of electricity annually by the year 2020, which equals to the annual output of about 50 brown power plants that have high carbon emissions [50][29]. To minimize the operational expenses and impacts on the environment, a variety of state-of-the-art works have been conducted to reduce data center energy consumption.

[3]See http://www.wikibench.eu/wiki/2007-10/ for more details.

There is a close relationship between resource utilization and energy consumption, as inefficient utilization of resource contributes to more power consumption [81]. Virtualization is an import technique in Clouds and it can improve resource utilization. Therefore, numerous energy-efficient resource scheduling approaches based on VM consolidation have been proposed. Consolidating VMs on fewer physical machines and turning the unused machines into the low-power mode reduce the number of active machines. Beloglazov et al. [33] proposed several VM consolidation algorithms to save data center energy consumption. The VM consolidation process has been modeled as a bin-packing problem, where VMs are regarded as items and hosts are regarded as bins. The objective of these VM consolidation algorithms is mapping the VMs to hosts in an energy-efficient manner. Based on the VM consolidation approaches in this work, other works like [34][43][71] have extended them to improve algorithm performance.

Mastroianni et al. [105] introduced a self-adaptive method for VM consolidation on both CPU and memory. The method aims to reduce the overall costs caused by energy-related issues. The VM consolidation process is determined by a probabilistic function, which is based on Bernoulli trial. Li et al. [91] developed a Bayesian network-based estimation model for VM consolidation and took nine data center factors into consideration. Zheng et al. [155] jointly considered VM consolidation and traffic consolidation together to minimize the servers and network energy consumption in data centers. The authors not only modeled the server power model, but also the switch model in the network. Experiments conducted under real environment showed that this joint approach can outperform the approaches that only adopt VM consolidation in terms of energy consumption and service delay. Habibi et al. [68] proposed a VM placement approach and routing system based on software defined network (SDN) and consider both energy consumption and QoS. Han et al. [70] presented a remaining utilization-aware algorithm for VM placement, and an energy-efficient algorithm to select hosts to shut down.

Another dominant approach to reduce energy consumption is Dynamic Voltage Frequency Scaling (DVFS). The DVFS approaches achieve energy reduction by adjusting frequencies of processors rather than using less active servers in VM consolidation. The DVFS approach investigates a trade-off between energy consumption and computing performance, where processors lower their frequency when they are lightly loaded and utilize full frequency when loads are heavy.

Kim et al. [83] modeled real-time service as real-time VM requests, and proposed several DVFS algorithms to reduce energy consumption for the DVFS-enabled cluster. Arroba et al. [24] proposed an approach combines DVFS and VM consolidation techniques by considering energy consumption and performance degradation together. Teng et al. [140] presented several heuristic algorithms combining DVFS and VM consolidation together for batch-oriented scenarios.

Some research taking both energy consumption and QoS into account have been conducted, which is also the consideration of our proposed software prototype system. Dou et al. [56] introduced an energy-aware dynamic VM scheduling approach for QoS enhancement in Clouds for big data, which aimed to benefit users with discount prices and reduce the execution time of tasks. Adhikary et al. [19] developed a QoS-aware and energy-aware cloud resource management system for multimedia applications, and proposed two resource management algorithms to reduce energy consumption while maintaining QoS of applications. Khanouche et al. [82] presented an energy-centered and QoS-aware service selection algorithm to deal with the multiple-objective problem for the Internet of Things applications. Wang et al. [147] proposed an energy-aware multi-dimension resource scheduling algorithm for cloud data centers to reduce energy and ensure QoS. Tang et al. [138] proposed a SLA-aware and resource-efficient algorithm for making auto-scaling policy, which aims to achieve energy saving and QoS improvement for VNFs. Cheng et al. [44] developed a power-aware resource provisioning approach, namely ePower for sustainable datacenter by considering renewable resources. The proposed approach aims to reduce system energy consumption via green power and fulfill QoS requirement under heterogeneous workloads.

VM consolidation and DVFS have been proven to be efficient to reduce energy consumption in both theory and practice, however, both of them cannot function well when the whole data center is overloaded. Thus, brownout is applied to handle data center overloads and reduce energy consumption. Klein et al. [85] applied brownout to design more robust applications under the overloaded or unpredicted situation. In our previous work, brownout was applied to save energy consumption in data centers. In Chapter 3, we presented the brownout enabled system model and proposed several heuristic policies to find the microservices or application components that should be deactivated for energy saving purpose. The results showed that a trade-off existed between energy consumption and discount, and in Chapter 4 , we adopted approximate Markov Decision Process to improve the trade-off.

Different from the energy-efficient approaches based on VMs, our software system is based on containers. Compared with VMs, containerization provides a fine-grained control on microservice resource usage. Kozhirbayev et al. [88] compared several existing container-based technologies for Clouds and evaluated their strength and weakness. They concluded that containers can give almost the same performance of native systems. Currently, most work related to containers are focused on the orchestration of containers construction and deployment [119]. For example, Liu et al. [95] proposed a flexible container-based computing platform for scientific workflow. Barresi et al. [28] introduced MicroCloud, which is a container-based solution for managing cloud resource efficiently. Kim et al. [84] proposed a virtual network function placement algorithm that allows users to meet QoS while minimizing energy consumption.

Maqsood et al. [103] proposed algorithms to reduce energy consumption and data access latency on single-chip cloud computers. Fang et al. [61] proposed a model based on predictive control approach to reduce data center power while maintaining QoS. The model was designed for a broader view of data centers, including the performance of hosts and thermal changes. Deng et al. [52] presented a mobile service selection approach for composition to reduce energy consumption in a mobile cloud environment. Bi et al. [36] introduced an approach to manage resources at the application level in cloud data centers, while they focused on economic profits perspective.

In Chapter 5, we have proposed an approach for managing energy in container-based clouds while focusing on scheduling algorithms design. Whereas, in this chapter, we focus on the design and development of a new software system supporting brownout-based energy-efficient management of clouds. To the best of our knowledge, our proposed software system is the first one to reduce energy consumption with brownout based on containers, which also considers both energy consumption and QoS.

6.3 System Architecture, Design and Implementation

The purpose of BrownoutCon is to provide a software system based on brownout and containers for energy-efficient cloud data centers. The system takes advantage of public APIs of Docker Swarm and is evaluated under Grid'5000 testbed. The system is designed to be extensible, which means new components can be added without the necessity to modify the original codes or configurations of Docker Swarm and Grid'5000.

Our software system is deployed on Docker Swarm master and worker nodes. Docker Swarm provides a platform for managing container cluster, monitoring status of swarm master and worker nodes, deploying containers on nodes, collecting resource usage of containers, controlling the lifecycle of containers, sending messages and commands between the master and worker nodes. Docker Swarm needs to be deployed on physical machines or virtual machines. Therefore, we adopt Grid'5000, a real testbed that provides access to ample resources for Docker Swarm deployment. We also take advantage of the Grid'5000 APIs to collect the energy consumption data of the machines. In the following sections, we discuss the system requirements, assumptions, system design and its implementation.

6.3.1 Requirements and Assumptions

The components of the proposed software prototype system are running on the Docker Swarm master and worker nodes. Our current implementation assumes that a single instance of each key components is invoked on the master node, such as components for controlling brownout, monitoring system status, managing deployment policies and managing models. On each worker node, a single instance of a component that collects node information is running. When new nodes are joining the Docker Swarm as worker nodes, the master node is responsible for deploying containers to the nodes.

BrownoutCon saves the energy consumption and handles overloads via temporarily disabling some containers, therefore, we assume that the services in the target system (e.g. e-commerce system) are implemented with microservice paradigm and some services (e.g. recommendation engine service) are not necessary to keep running all the time.

The main optimization objective of our software system is reducing energy consumption, a precise power probe to collect energy usage is required. Container scheduling policies may use the energy usage data to make decisions on controlling containers.

Another requirement is that a manager is needed to control all the hosts to turn them into low-power mode or active. This manager is used by the brownout controller on master node to connect with other worker nodes via the communication protocol. Grid'5000 has provided the APIs to switch the status of hosts, and more details will be introduced in the following sections.

6.3.2 BrownoutCon Architecture

The architecture of BrownoutCon is same as the architecture depicted in Figure 5.1. We briefly introduce the main components as below:

1) **Users:** This component contains user and requests information. It also captures system configurations such as predefined QoS constraints (e.g. average response time and SLA violations), energy budget and service deployment patterns according to users' demand.

2) **Cloud Service Repository:** This component manages the services offered to users, including service information, such as service name and image. Each service may be constructed via a set of microservices. In order to manage microservices with brownout, the microservices are identified as mandatory or optional.

a. Mandatory Microservices: These microservices keep running all the time when they are started and cannot be temporarily stopped, like database related microservices.

b. Optional Microservices: These microservices can be deactivated temporarily depending on system status. Microservices are connected if there are communications between them. We consider that if one optional microservice is deactivated, then other connected microservices should also be deactivated.

Notes: A microservice can be identified as optional if the service/content it provides is defined as optional by its creators. For instance, the online recommendation engine in the online shopping system and the spell checker in the online editor system can be identified as optional microservices under resource constrained situations.

3) Execution Environment: This component provides the container-based environment for microservices or containers. The dominant execution environments for microservices or containers are Docker, Kubernetes, and Mesos. In BrownoutCon, we use Docker as the execution environment for microservices.

4) Brownout Controller: This component controls optional microservices or containers based on system status . It applies policies introduced in Section 6.4 to provide an efficient solution for managing brownout and containers. As mentioned in Section 6.1, brownout has a control knob called dimmer that represents the probability to execute microservices. We make some adjustments to make the dimmer of brownout to be adapted to this component as well as our architecture. Our dimmer is only applied to the optional microservices and its value is computed according to the severity of system overloads (the number of overloaded hosts in the data center).

5) System Monitor: It is a component that monitors the health of nodes and collects hosts resource consumption status. It uses the third-party toolkit to support its function, such as Grid'5000 public APIs that provide real-time data on infrastructure metrics, including host health, CPU utilization, and power consumption.

6) Scheduling Policy Manager: This component provides and manages the policies for Brownout Controller to schedule microservices/containers. In order to ensure the energy budget and QoS constraints, different policies designed for different preferences are required. For instance, if the service provider wants to balance the trade-off between energy and QoS, then a policy that considers the trade-off is preferred.

7) Models Management: This component maintains the energy consumption and QoS models in the system. In BrownoutCon, the power consumption model is closely related to the utilization of microservice or container, and the QoS model is applied to define the QoS constraints.

8) Cloud Infrastructure: Under Infrastructure as a Service model, it is a component that offers physical resources to users, where microservices or containers are deployed. In our experiments, we use Grid'5000 as our infrastructure. More details are given in Section 6.5.

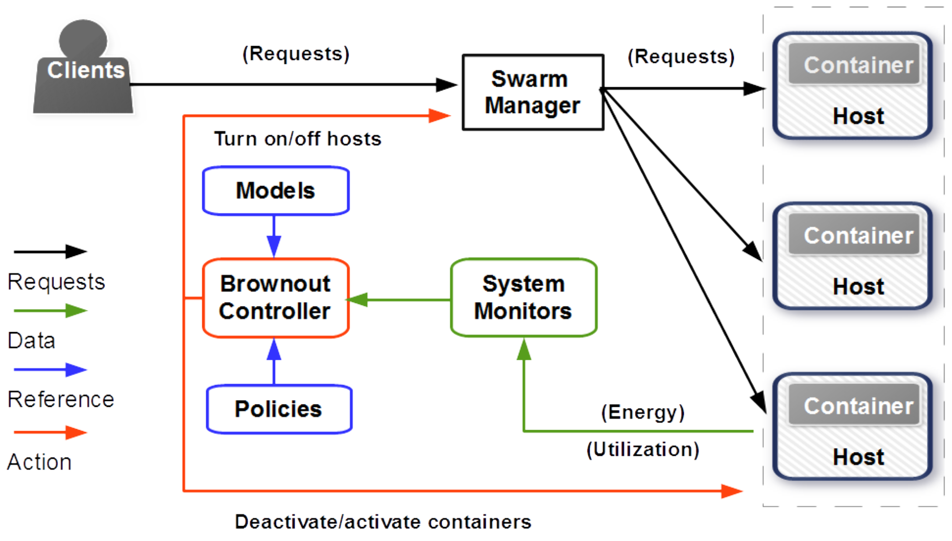


Figure 6.1: Energy-efficient scheduling architecture

6.3.3 Energy-efficient Scheduling Architecture

The main purpose of our software system is energy efficiency, and the main approach to achieve this goal is through energy-efficient scheduling policies. Deriving from Brownout-Con architecture, Figure 6.1 shows the energy-efficient scheduling architecture based on brownout, which depicts the BrownoutCon from the energy-efficient scheduling perspective.

In this scheduling architecture, clients submit their requests to the system, and Docker Swarm Manager dispatches the requests to containers and hosts. The System Monitors collect the energy and utilization information from hosts, and then send the information to Brownout Controller. With the information from System Monitors, the Brownout Controller refers to the host power consumption or utilization models to compute how much utilization/energy should be reduced. Then the Brownout Controller makes decisions based on scheduling policies to switch the states of hosts and containers, such as turning the hosts into low-power mode or deactivating containers.

6.3.4 Integration with Docker Swarm

BrownoutCon is installed on Docker Swarm node independently of Docker Swarm services. In addition, the activities of BrownoutCon are transparent to the Docker Swarm

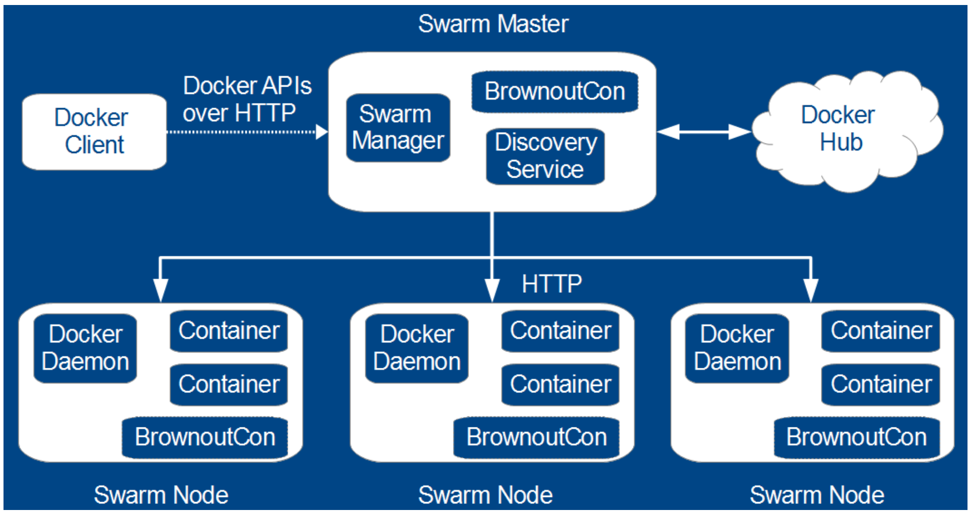


Figure 6.2: BrownoutCon integrated with Docker Swarm

services, which means Docker Swarm does not need to reconfigure to fit with BrownoutCon and use its brownout feature. In other words, BrownoutCon can be installed on existing Docker Swarm cluster without modifying the configurations.

BrownoutCon achieves the transparency via the interactions with the public APIs of Docker Swarm cluster. BrownoutCon uses the APIs to obtain information about containers deployment, containers utilization, and containers properties. Although the operations of BrownoutCon will affect the system status and containers state by deactivating or activating containers, it is transparently processed by Docker Swarm public APIs.

The implication of this integration approach represents that the container deployment is handled by Docker Swarm, and BrownoutCon makes decisions on deactivation or activation of containers. Figure 6.2 shows how BrownoutCon is integrated into Docker Swarm. In Docker Swarm, the nodes are categorized as two classes: swarm master node and swarm worker node. The master node is responsible for maintaining cluster state, scheduling services (containers) and serving swarm mode with Docker APIs over HTTP, while the purpose of worker nodes is executing containers. The respective BrownoutCon components are deployed on master and worker nodes.

6.3.5 Containers Deployment with Compose File

Docker provides compose[4] tool to deploy multiple containers, in which a configuration file is used to configure containers properties. With the compose file, the containers can be easily deployed and managed on clusters. In the compose file of our web application, to identify the recommendation engine microservice as optional, we labeled it as optional in the brownout feature. Moreover, as previously mentioned, the optional containers are only allowed to be deployed on the worker node, thus, we configure the placement constraint of this microservice as the worker node. More deployment properties can also be configured in the compose file.

6.3.6 Entity Interaction Diagram

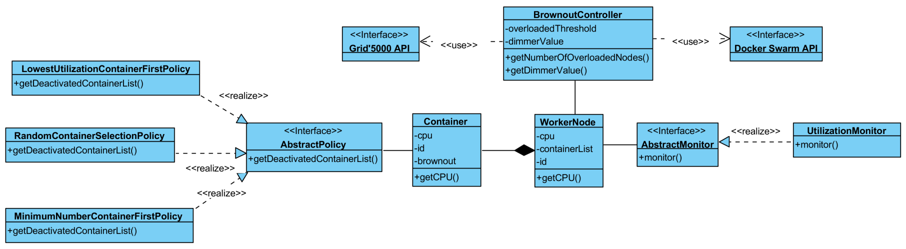


Figure 6.3: Entity interactions in BrownoutCon

To implement the aforementioned architecture and functionalities, we use Java to develop our software system. The main classes of BrownoutCon are depicted in Figure 6.3. The details of these classes are as below:

Docker Swarm API: This class wraps Docker Swam APIs and provides the interface for BrownoutController class to call. The Docker Swarm APIs offer the functions to fetch the information of containers and operate on containers, such as collecting containers utilization, containers id, containers property (optional or mandatory), deploying and updating containers with the compose file, deactivating and activating containers.

Grid'5000 API: This class uses Grid'5000 APIs to collect hosts energy consumption and switch status of hosts. Grid'5000 provides APIs to gather the total power at per second rate for all the hosts in data center. The APIs also allow BrownoutController class to switch the hosts into low power mode or turn the hosts on.

WorkerNode: This class models the host in the data center. Attributes of a Work-

[4]See https://docs.docker.com/compose/compose-file/ for more details.

erNode include CPU utilization and the containers deployed on the host. The software system initializes a WorkerNode instance for each host. To be consistent with the status of real hosts, when the software system is running, the WorkerNode instances will keep updating their CPU utilization and container lists .

AbstractMonitor: It provides an interface to monitor the status system. The class UtilizationMonitor implements the AbstractMonitor interface and focuses on monitoring hosts utilization. With the monitored information, the system can know how many hosts are overloaded and make decisions based on this information. Other monitors, such as memory or network monitors can be extended if they implement the AbstractMonitor.

Container: The Container class models the containers deployed on hosts. The class defines the basic information of containers, including container id, CPU utilization and other information that can be fetched via Docker Swarm APIs. To apply brownout on containers, the Container class also has the attribute to identify whether a container is optional or mandatory.

AbtractPolicy: It is an interface that defines the functions that scheduling policies should implement. To deactivate some containers temporarily and reduce energy consumption, the policies that implement the AbstractPolicy interface are responsible for finding the containers that should be deactivated. More scheduling policies can be implemented only if they implement the AbstractPolicy interface. The details of our implemented policies in BrownoutCon will be introduced in Section 6.4.

BrownoutController: This class is the core class of our software system. It assembles all the information from different sources and makes the decision for controlling hosts and the containers on them. The overloaded threshold is defined in BrownoutController class to determine whether a host is overloaded. BrownoutController knows system status from Docker Swarm APIs, Grid'5000 APIs and WorkerNode instances, and triggers brownout to handle overloads and reduce energy consumption via operations on hosts or containers.

6.3.7 Sequence Diagram

To provide an in-depth understanding of the working process of BrownoutCon, Figure 6.4 shows a sequence diagram of handling requests by our software system. Firstly, the

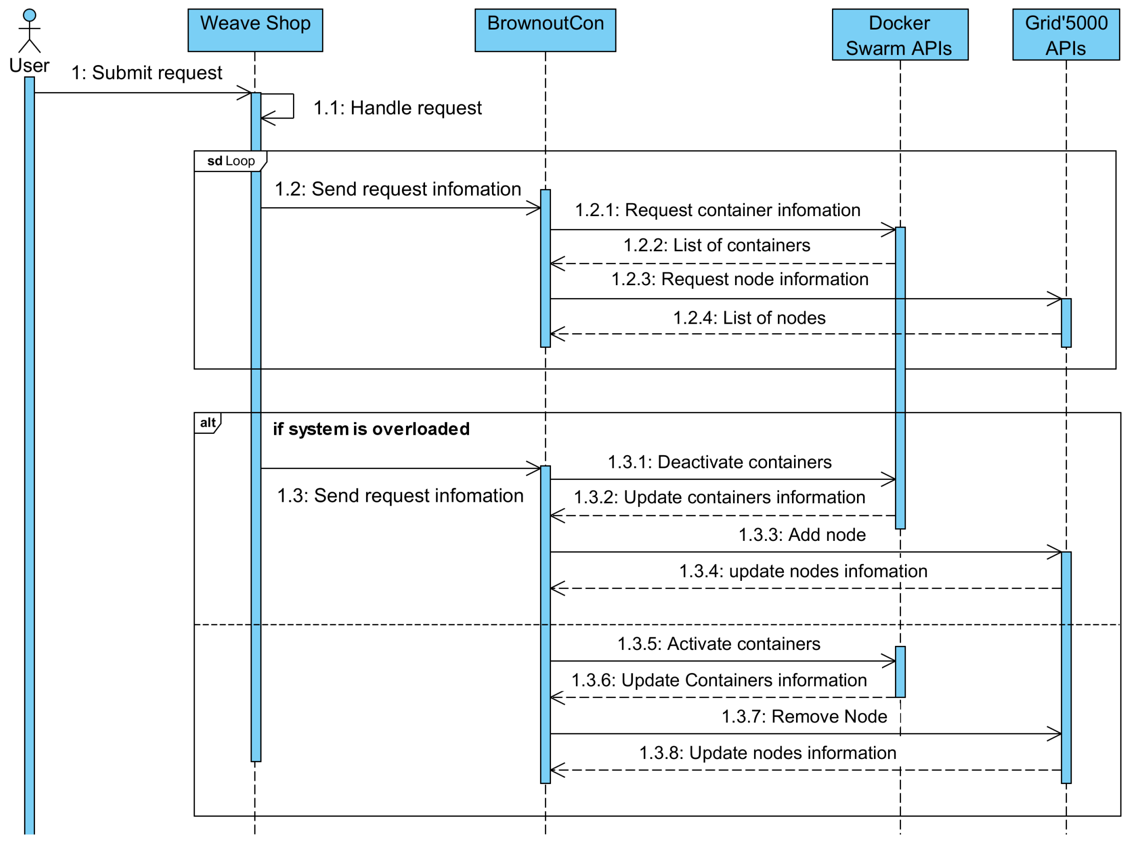

User
Weave Shop
BrownoutCon
Docker Swarm APIs
Grid'5000 APIs
1: Submit request
1.1: Handle request
sd Loop
1.2: Send request infomation
1.2.1: Request container infomation
1.2.2: List of containers
1.2.3: Request node information
1.2.4: List of nodes
alt
if system is overloaded
1.3: Send request infomation
1.3.1: Deactivate containers
1.3.2: Update containers information
1.3.3: Add node
1.3.4: update nodes infomation
1.3.5: Activate containers
1.3.6: Update Containers information
1.3.7: Remove Node
1.3.8: Update nodes information

Figure 6.4: Sequence diagram of of handling requests by BrownoutCon

users submit their requests to a web application called Weave Shop (more details about this application will be introduced in Section 6.5.2.) Then the Weave Shop sends the information of requests to BrownoutCon, and BrownoutCon keeps collecting nodes and containers information periodically via Grid'5000 and Docker Swam public APIs respectively. When BrownoutCon is collecting information, if the system is overloaded, which means the Weave Shop cannot handle all the incoming requests, the BrownoutCon adds nodes to serve requests. The BrownoutCon also triggers brownout-based policies to deactivate containers to relieve overloads and reduce energy consumption. After these operations, the information of the nodes and containers are updated. Once the system is not overloaded, BrownoutCon activates the containers or removes the nodes from active nodes list (switching nodes into low-power mode). Upon the completion of operations on containers and nodes, the updated information is sent to BrownoutCon.

6.3.8 Problem Formulation

One of the objectives of BrownoutCon is reducing energy consumption. The total energy consumption $E(t)$ during time interval t is formed as the sum of all the host energy consumption in the data center as shown in Equation 6.1. Here, we only care about the physical server's energy consumption rather than other network devices or cooling equipment.

$$E(t) = \sum_{i=0}^{n-1} \int_{t}^{t+1} P_i(t)dt \tag{6.1}$$

where n is the total number of hosts in the data center, and $P_i(t)$ is the power at time t of host i. The energy consumption of each physical server in Grid'5000 can be collected via APIs.

Another objective of BrownoutCon is ensuring the QoS, thus, to quantify the overall QoS of the system, we use the average response time to measure the time between sending a request and receiving the response. We also use SLA Violation Ratio (SVR) to quantify the failed requests, which is formalized as:

$$SVR = \frac{num_{err}}{num_a} \tag{6.2}$$

where num_a is the total number of requests sent to the system, and num_{err} is the number of requests failing to get the response.

The optimization objective of BrownoutCon is minimizing the total power consumption while satisfying QoS. Thus, the optimization problem of resource scheduling in BrownoutCon can be formulated as:

$$\begin{aligned} min \sum_{t=0}^{T} E(t) \\ s.t. R_{avg} \leq \alpha, \forall T \\ SVR \leq \beta, \forall T \end{aligned} \tag{6.3}$$

where $\sum_{t=0}^{T} E(t)$ is the total power consumption in the data center during the observation time period T, R_{avg} is the average response time and α is the allowed average response time defined by the service provider, and β is the allowed SLA violation ratio.

6.4 Policies Implemented in BrownoutCon

BrownoutCon has implemented several policies that proposed in Chapter 3 and the VM consolidation aspect is excluded. In Chapter 3, the policies were evaluated through simulations in CloudSim. As introduced in Section 6.1, the scheduling problem can be divided into several sub-problems: (1) workload prediction; (2) overloaded status detection (3) brownout trigger; (4) deactivated containers selection; and (5) hosts scaling. In this section, we will introduce the implemented policies for reference. It is noted that the introduced policies are not the main focus of this chapter. The focus of this work is designing and implementing the software system based on brownout and containers.

6.4.1 Workload Prediction

To predict the future workloads based on the previous workloads, we adopt the sliding windows as shown in Algorithm 6.1. The motivation of sliding windows is giving more weights to the requests rates of recent time intervals. Let L_w to be the window size, $num(t)$ to be the number of requests at time interval t, we estimate the number of requests for the next time interval $t+1$ as in Equation 6.4. And the sliding window is moving forward along with the time.

Algorithm 6.1: Algorithm for predicting future workload based on sliding windows

```
Input  : sliding window size $L_w$, the number of requests at previous $L_w$ time intervals, the
         predicted time interval $t$ ($t \geq L_w$)
Output: the predicted number of requests $n\hat{u}m(t)$ at time interval $t$
1 for $k$ from $t - L_w$ to $t - 1$ do
2     $n\hat{u}m(k+1) \leftarrow n\hat{u}m(k) + num(k)$
3 end
4 $n\hat{u}m(t) \leftarrow n\hat{u}m(k+1)/L_w$
5 return $n\hat{u}m(t)$
```

$$n\hat{u}m(t) = \frac{1}{L_w} \sum_{k=t-L_w}^{t-1} num(k) \tag{6.4}$$

6.4.2 Overloaded Host Detection

In our experiments, we use a predefined overloaded threshold to detect whether a host is overloaded or not. For instance, if the overloaded threshold is defined as 85%, the host is regarded as overloaded when its CPU utilization is above 85%. Currently, we only adopt CPU utilization to detect the overloaded host. Algorithm 6.2 shows the pseudocode of our overloaded host detection algorithm.

Algorithm 6.2: Algorithm for detecting the number of overloaded hosts

```
Input  : overloaded threshold $T_u$, the number of hosts $n$ in data center, CPU utilization $u_i$ of host $i$
Output: the number of overloaded hosts $n_o$
1 for $i \leftarrow 0$ to $n-1$ do
2     if $u_i \geq T_u$ then
3         $n_o \leftarrow n_o + 1$
4     end
5 end
6 return $n_o$
```

Equations 6.5 and 6.6 show the way to calculate the number of the overloaded host. We use n_i^o to denote whether host i is overloaded or not, which is detected by the utilization u_i and overloaded threshold T_u. If u_i is no less than T_u, n_i^o equals to 1, otherwise it equals to 0. The total number of the overloaded host is denoted as n_o, which is the sum of n_i^o for all the hosts.

$$n_i^o = \begin{cases} 1, \ if \ u_i \geq T_u \\ 0, \ if \ u_i < T_u \end{cases} \tag{6.5}$$

$$n_o = \sum_{i=0}^{n-1} n_i^o \tag{6.6}$$

6.4.3 Brownout Trigger

Once there are hosts detected as overloaded, the brownout mechanism will be triggered to handle the overloads as well as to reduce energy consumption. As mentioned in Section 6.1, firstly, the algorithm is required to calculate the dimmer value, which is the control knob to represent the probability to trigger brownout on hosts.

The pseudocode of brownout trigger algorithm is presented in Algorithm 6.3, the dimmer value θ_t is calculated based on the number of overloaded hosts (line 1). The dimmer value θ_t at time t is calculated based on the number of overloaded hosts n_o as shown in Equation 6.7:

$$\theta_t = \sqrt{n_o/n} \tag{6.7}$$

Then the algorithm computes the expected utilization reduction on overloaded hosts. The expected utilization reduction u_i^r of host i is the product of dimmer value θ_t and the host utilization u_i as:

$$u_i^r = \theta_t \times u_i \tag{6.8}$$

Algorithm 6.3: Brownout trigger algorithm

```
Input  : the overloaded threshold $T_u$, the number of overloaded hosts $n_o$, the number of hosts $n$ in
         data center, dimmer value $\theta_t$ at time $t$, CPU utilization $u_i$ of host $i$
Output: expected utilization reduction list $u_i^r$
1 $\theta_t \leftarrow \sqrt{n_o/n}$
2 for $i \leftarrow 0$ to $n-1$ do
3     if $u_i > T_u$ then
4         $u_i^r \leftarrow \theta_t \times u_i$
5     end
6 end
7 return $u_i^r$
```

6.4.4 Deactivated Containers Selection

Based on the expected utilization reduction, the policies select containers to deactivate based on different containers selection policies. In BrownoutCon, we have implemented three containers selection policies for deactivation. Based on the strategy design pattern[5], these policies implement the AbstractPolicy interface in Figure 6.3 and can be selected independently at runtime.

1) Lowest Utilization Container First policy

The Lowest Utilization Container First (LUCF) policy selects a set of containers to reduce the utilization of overloaded hosts. The objective of LUCF is that the utilization after reduction is expected to be less than the overloaded threshold, and the difference between the expected utilization reduction and the sum of deactivated containers utilization is

[5]See https://en.wikipedia.org/wiki/Strategy_pattern for more details.

minimized. Thus, the host utilization is reduced and the reduced utilization is close to the expected reduction. The deactivated container list is defined in Equation 6.9. We use $u_i^{'}$ to denote the utilization of host i after the containers in the deactivated lists are deactivated, which equals to $u_i - u_i^{dcl}$. The utilization of all the containers in dcl_i is denoted as u_i^{dcl}. The $\min(|u_i^r - u_i^{dcl}|)$ represents the to minimize the absolute value of $u_i - u_i^{dcl}$.

$$dcl_i = \begin{cases} \{u_i^{'} \leq T_u, \min(|u_i^r - u_i^{dcl}|)\}, & if\ u_i \geq T_u \\ \emptyset, & if\ u_i < T_u \end{cases} \tag{6.9}$$

Algorithm 6.4 presents the pseudocode of LUCF. The LUCF sorts the optional containers list ocl_i based on container utilization in ascending order so that the container with the lowest utilization is at the head of the list. The size of ocl_i is $ocl_i.size()$. The algorithm checks the hosts one by one, if the first container c_0 on host i has the utilization greater than u_i^r, c_0 is put into the deactivated container list dcl_i. Since we consider connected microservices, the policy also adds the container's connection tag T_0^c (a string value) that indicates how it is connected with other containers into a set S for recording connections. However, if the utilization of the first container is less than the expected utilization reduction, LUCF finds a containers sublist to deactivate more containers. The sublist is the one that has the sum of utilization that is closest to the expected utilization reduction than other sublists. Same as previous operations, these containers are put into the deactivated container list dcl_i and their connection tags are put into the set S. Then, the algorithm finds other connected containers and put them into the deactivated container list.

2) Minimum Number of Containers First policy

As formalized in Equation 6.10, in order to deactivate fewer containers so that more optional functionalities can be provided, we also implement Minimum Number Containers First (MNCF) policy, which selects the minimum number of containers while saving the power consumption. Since it is quite similar to the LUCF, the pseudocode of MNCF is not provided here. The $\min(dcl_i.size())$ represents the objective to minimize the size of the deactivated container list.

Algorithm 6.4: Lowest Utilization Container First policy (LUCF)

```
Input  : the number of hosts $n$ in data center, overloaded threshold $T_u$, deactivated container list
         $dcl_i$ on host $i$, the optional container list $ocl_i$ of host $i$, which is sorted based on utilization
         of containers $u_j^c$ in ascending order, deactivated tag set $S$, connection tag $T_j^c$ of container $c_j$
Output: deactivated container list $dcl_i$
for $i \leftarrow 0$ to $n-1$ do
    if $u_i > T_u$ then
        if $u_0^c \geq u_i^r$ then
            add $c_0$ into $dcl_i$
            add $T_0^c$ into $S$
        end
        for $c_j$ in $ocl_i$ ($j = 0, 1, 2, \dots, ocl_i.size() - 1$) do
            if $(u_j^c \leq u_i^r)$ & $(u_i^{dcl} \leq u_i^r)$ then
                add $c_j$ into $dcl_i$
                add $T_j^c$ into $S$
                minimize $|u_i^r - u_i^{dcl}|$
            end
        end
        forall $c_j$ in $ocl_i$ ($j = 0, 1, 2, \dots, ocl_i.size() - 1$) do
            if $T_j^c$ in $S$ then
                add $c_j$ into $dcl_i$
            end
        end
    end
    deactivate containers in $dcl_i$
end
return $dcl_i$
```

$$dcl_i = \begin{cases} \{u_i' \leq T_u, \min(dcl_i.size())\}, & if\ u_i \geq T_u \\ \emptyset, & if\ u_i < T_u \end{cases} \tag{6.10}$$

3) Random Container Selection policy

Based on a uniformly distributed discrete random variable X that selects a subset of dcl_i randomly, the Random Container Selection (RCS) policy uses uniform distribution function $U(0, ocl_i.size() - 1)\}$ to randomly select a number of optional containers to reduce energy consumption, as presented in Equation 6.11.

$$dcl_i = \begin{cases} \{u_i' \leq T_u, X = U(0, ocl_i.size() - 1)\}, & if\ u_i \geq T_u \\ \emptyset, & if\ u_i < T_u \end{cases} \tag{6.11}$$

6.4.5 Hosts Scaling

To scale the number of active hosts, we adopt the hosts scaling algorithm in [143] as shown in Algorithm 6.5, which is a predefined threshold-based approach. With profiling experiments, we set the overloaded requests threshold as the number of requests when the host cannot respond within an acceptable time limit. The algorithm computes the required hosts as the predicted number of request divided by the profiling number of requests of the overloaded threshold. If the required number of hosts is more than current active hosts, more hosts will be added to provide services, otherwise, if current active hosts are adequate, then the excess machine can be set as low-power mode to save energy consumption.

Algorithm 6.5: Hosts scaling algorithm

```
Input  : number of hosts $n$ in data center, number of active hosts $n_a$, number of requests when host
         is overloaded $num_{thr}$, predicted number of requests $\hat{num}(t)$ at time $t$.
Output: number of active hosts $n_a$
1  $n_a \leftarrow \lceil \hat{num}(t) \div num_{thr} \rceil$
2  $n' \leftarrow n_a - n$
3  if $n' > 0$ then
4      Add $n' hosts$
5  else if $n' < 0$ then
6      Remove $|n'|$ hosts
7  else
8      no host scaling
9  end
10 return $n_a$
```

6.5 Performance Evaluation

In this section, we evaluate our proposed software prototype system by conducting experiments under Grid'5000 infrastructure. The goals of this section include: 1) evaluating the behavior of the software system in a test environment; 2) showing the availability of the proposed system so that fostering and facilitating research efforts and future work in brownout and container-based area.

6.5.1 Workload Traces

To make the experiments reproducible, we use the real trace from Wikipedia requests on 2007 October 17 to replay the workload of Wikipedia users. The trace includes data on requests time stamp and their accessed pages. We filter[6] the requests based on per second rate and generate the requests rate. The original request rate is around 1,500-3,000 per second. To scale the workload set to fit with our experiments, we use 10% of the original user requests size. Figure 6.5 shows the requests rate per second during the day. The blue line is the actual trace derived from Wikipedia and the red line is the predicted trace based on the sliding window (sliding window size is 5) as introduced in Section 6.4. We can observe some anomalies during intervals 600-800, which can be due to the unpredicted network congestion. While during most time intervals, the variances between actual trace and predicted trace are small. This means the predicted trace can serve a guide for host scaling strategy.

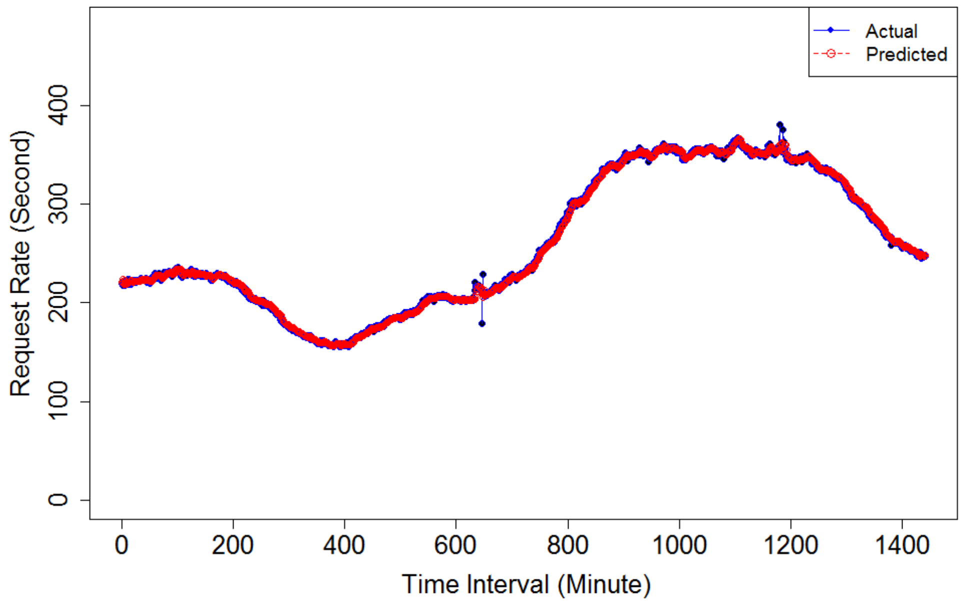


Figure 6.5: Requests rate of Wikipedia trace

[6]The details about how we filter the raw data are provided as supplementary materials.

6.5.2 Application Deployment

We use the Weave Shop[7] web application that implemented with containers as the application in our scenario. The Weave Shop is a shopping system for selling socks online and has multiple microservices, including user microservice to handle user login, user database microservice for user information storage, payment microservice to process transactions, font-end microservice to show the user interface, catalog microservice for managing item for sale and etc. As these microservices are implemented independently, they can be deployed and controlled without impacting other microservices. The application is deployed by the compose file as introduced in Section 6.3.5, and part of the microservices are configured as optional, e.g. recommendation engine is noted as optional.

The user interface may be influenced due to the deactivation of some microservices. Figure 6.6 shows the user interface of Weave Shop application. Figure 6.6a is the user interface when full services are provided during no resource saturated scenario, while Figure 6.6b illustrates the user interface when brownout is triggered and the recommendation engine service/container is deactivated. As a result, other recommended products are not showed in Figure 6.6b

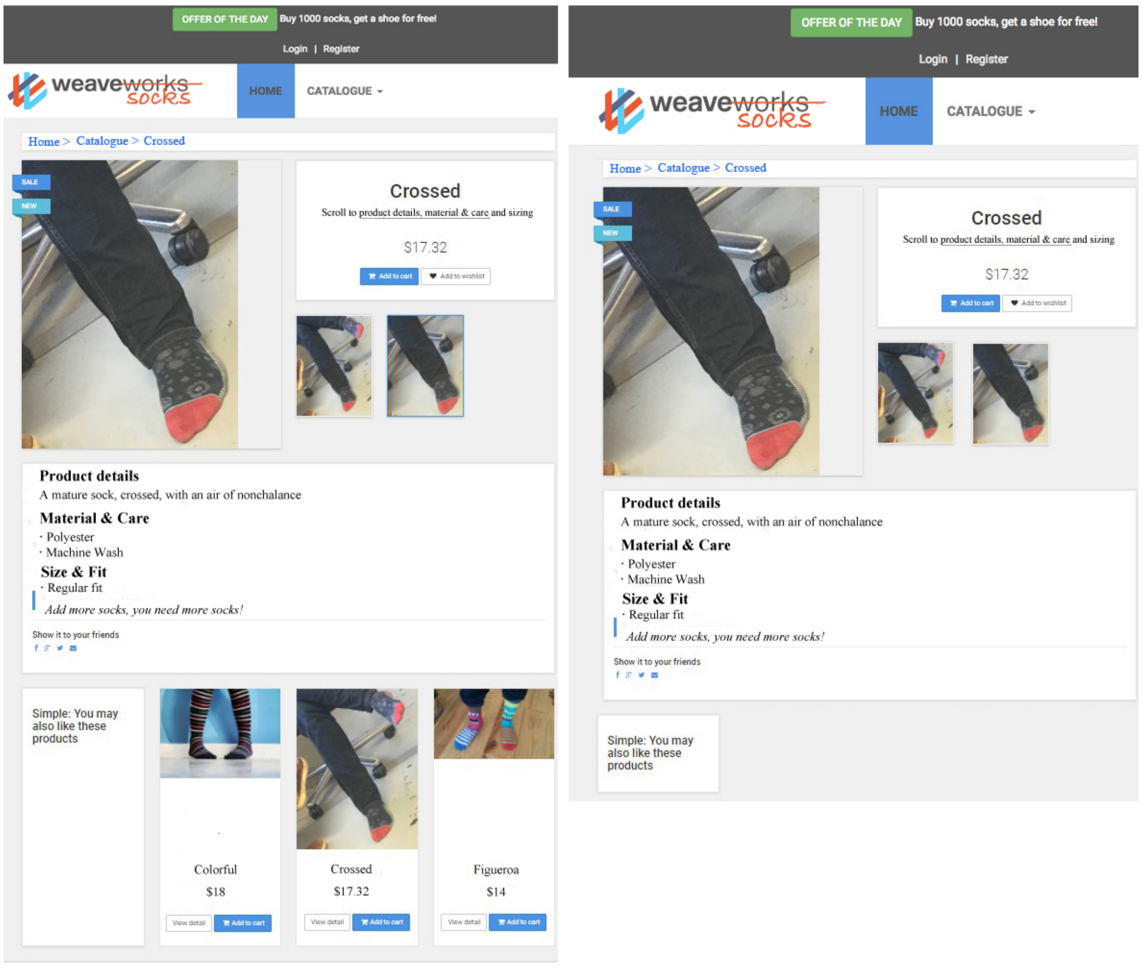
6.5.3 Experimental Testbed

The testbed we used for evaluation is Grid'5000, and we adopt the cluster equipped with power measurement APIs at Lyon site. The hardware specifications are as below:

- 11 $\times$ Sun Fire V20z[8] with AMD Opteron 250 CPU (2 cores, 2.4GHz) and 2GB memory, and all the hosts are running on Debian Linux operating system.

We choose the machines in the same site so that we can reduce the network influence of uncontrolled traffics from other sites. Among the 11 servers, nine are running as the Docker Swarm worker nodes, one is running as the Docker Swarm master node, and another node contains workload trace and installed with JMeter[9] for sending requests to Docker Swarm cluster. The energy consumption of the workload trace node is not

[7] See https://github.com/microservices-demo/microservices-demo for more details.
[8] The maximum power of this model is 237 Watts, and the sleep mode consumes 10 Watts.
[9] See http://jmeter.apache.org/ for more details.



(a) Brownout is not triggered

(b) Brownout is triggered

Figure 6.6: Impact on user interface when brownout is triggered

counted, as it is not regarded as a part of the cluster to provide services. All required softwares, such as Docker, Java, Ansible[10] and JMeter have been installed on these machines to minimize the effects of CPU utilization and network delay.

6.5.4 Experimental Design and Setup

In our experiments, the overloaded threshold is configured as 85%, and other parameters of BrownoutCon are configured as default except for the optional utilization percentage. The optional utilization percentage is configured as how much CPU utilization is allowed

[10]See https://www.ansible.com/ for more details.

to be given to optional containers. According to Chapter 3 , the change of this parameter has an impact on energy consumption. This parameter is varied for each experiment by going through the algorithms as below:

1. NPA algorithm [33] — A baseline algorithm that does not consider overloads and optional containers, where the hosts are running all the time and containers are not deactivated.

2. HS [143] — Another baseline algorithm that applies the host scaling algorithm in Algorithm 6.5, while not applying brownout-based policies.

3. The LUCF, MNCF and RCS algorithms introduced in Section 6.4 are with the varied optional utilization percentages from 10% to 30% in increments of 10%.

We evaluate the energy consumption, average response time and SLA violation ratio for these algorithms. We run each experiment three times to deal with the variance resulted from random factors, such as initial containers deployment, network latency, and application running status.

6.5.5 Experimental Results and Analysis

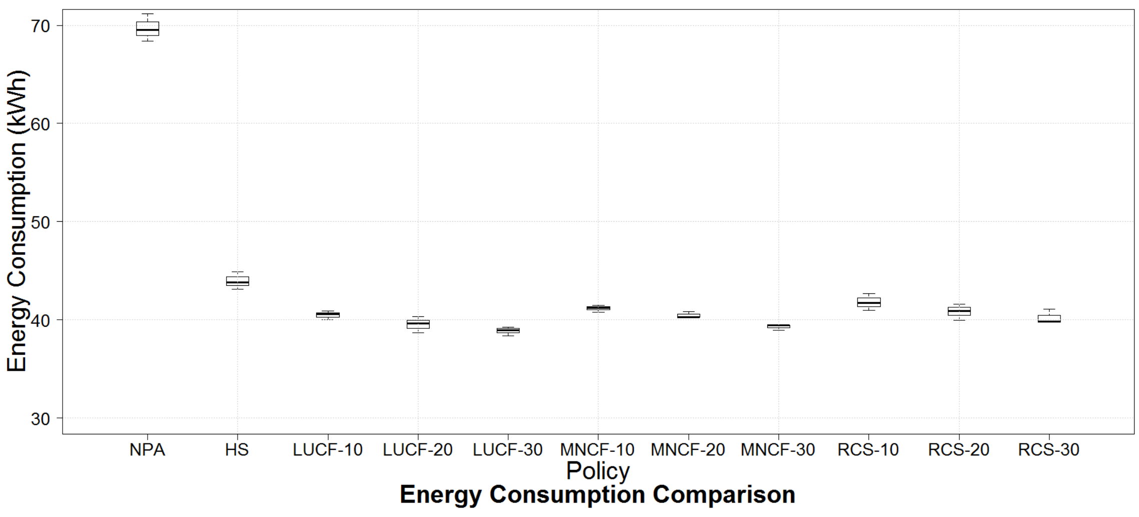


Figure 6.7: Energy consumption comparison

Figure 6.7 depicts the energy consumption comparison of different algorithms. From the results, NPA has the highest energy consumption with 69.6 kWh, and HS reduces it to 43.9 kWh. For the brownout-based algorithms with varied parameters, the energy consumption of LUCF is from 40.5 kWh to 38.8 kWh when the optional utilization percentage is increased from 10% to 30%. For the MNCF, its energy is close to LUCF, which ranges from 41.1 kWh to 39.2 kWh when optional utilization percentage is varied. As for the RCS, it decreases the energy from 41.4 kWh to 38.8 kWh. All the brownout-based algorithm have shown a significant reduction in energy consumption than NPA, and they can also save about 6% to 12% power consumption than HS with different parameter settings.

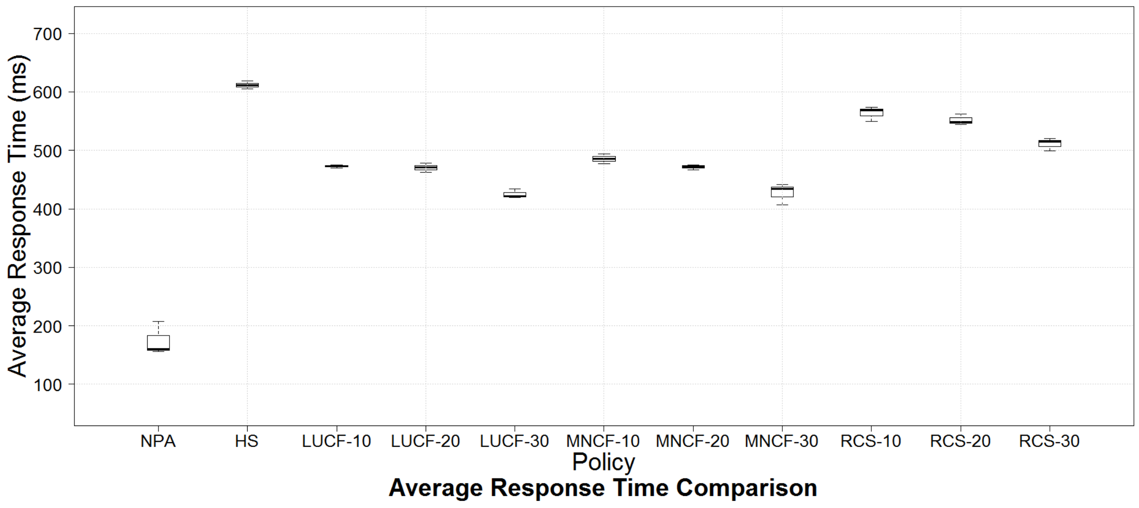


Figure 6.8: Average response time comparison

We also compare the average response time in Figure 6.8. Although NPA consumes more energy than other algorithms, with the adequate resources, the average response time is the lowest as 174.3 ms. The average response time of HS is 611.6ms, while the other brownout-based algorithms decrease this value. LUCF lowers the average response time from 472.6 ms to 425 ms, MNCF reduces it from 485.6 ms and reaches 427.3 ms with 30% optional utilization percentage, and the average response time of RCS ranges from 564.0 ms to 511.3 ms.

In Figure 6.9, the SLA violation ratios are compared. As NPA has enough resources, it does not experience any SLA violation, while in HS, it has 4.3% SLA violation. Compared with HS, LUCF relieves the SLA violated situation, reducing it from 2.1% to 0.5%. Similar to LUCF, MNCF also decreases the SLA violation to 0.5% from 2.3% when more optional utilization percentage is allowed. As for RCS, its SLA violation drops from 3.2% to 0.9%.

The mean values of obtained results are also displayed in Table 6.1. HS saves more

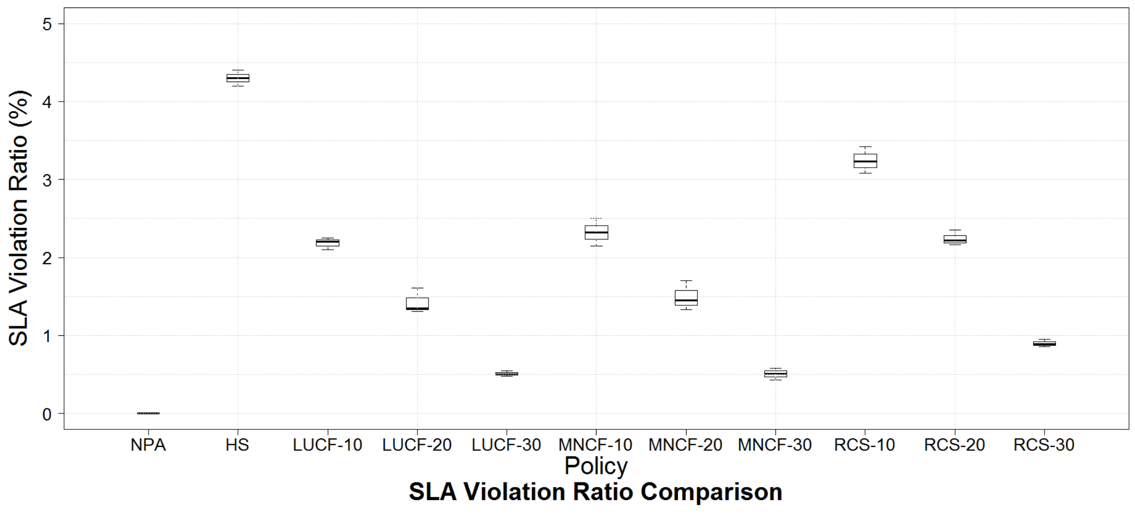


Figure 6.9: SLA violation ratio comparison

Table 6.1: The experiment results

Algorithm	Energy	Avg. Response Time	SLAVR
NPA	69.6 kWh	**174.3 ms**	-
HS	43.9 kWh	611.6 ms	4.3 %
LUCF-10	40.5 kWh	472.6 ms	2.1 %
LUCF-20	39.5 kWh	470.3 ms	1.4 %
LUCF-30	**38.8 kWh**	425.0 ms	**0.5 %**
MNCF-10	41.1 kWh	485.6 ms	2.3 %
MNCF-20	40.4 kWh	471.3 ms	1.4 %
MNCF-30	39.2 kWh	427.3 ms	0.5 %
RCS-10	41.4 kWh	564.0 ms	3.2 %
RCS-20	39.8 kWh	551.6 ms	2.2 %
RCS-30	38.8 kWh	511.3 ms	0.9 %

energy than NPA because it dynamically turns hosts into low-power mode, however, since resources are limited and without brownout, HS also experiences higher average response time and SLA violation ratio. We can also conclude, the brownout-based algorithms, LUCF, MNCF, and RCS save more energy than NPA and HS while ensuring QoS by reducing average response time and SLA violation ratio. The reason lies in that the brownout-based algorithms reduce energy by deactivating a set of containers and improve the QoS compared with the overloaded situation. And the performance differences of brownout-based algorithms are due to the different selections of deactivated containers. For the comparison of brownout-based algorithms, when more optional utilization percentage is provided, the algorithms perform better. Therefore, in practice, our software system works better if more containers are allowed to be deactivated, which also means that better performance of brownout-based approach can be achieved when more containers are configured as optional.

6.5.6 Scalability Discussion

The design and implementation of BrownoutCon are based on Docker Swarm, therefore, the performance of BrownoutCon is relevant to the scalability of Docker Swarm. Scalability tests on Docker Swarm have been conducted in [100] with 1000 nodes and 30,000 containers, which shows that Docker Swarm is highly scalable system.

6.6 Summary

In this chapter, we proposed the design and development of a software system based on brownout and containers for energy-efficient clouds, called BrownoutCon. BrownoutCon is transparent system based on Docker Swarm for containers management and does not require to modify the default configurations of Docker Swarm via using its APIs. BrownoutCon can be customized for implementing brownout-based algorithms, which dynamically activates or deactivates containers to handle overloads and reduce energy consumption. The experiments conducted on Grid'5000 infrastructure show that the brownout-based algorithms in BrownoutCon are able to reduce energy consumption while ensuring QoS. The proposed software can be applied in the container-based environment as well as future research in brownout area.

Renewable energy provides another promising direction to address the concern of energy efficiency problems in cloud computing environments. In the next chapter, we will present a self-adaptive approach for sustainable cloud data centers by managing applications and renewable energy.

Chapter 7

Approach for Managing Applications and Harnessing Renewable Energy

The attractive features like elasticity, availability and pay-as-you-go pricing model contributed towards rapid adoption of cloud computing. However, the huge energy consumed by cloud data centers makes them to be one of the fastest growing sources of carbon emissions. Approaches for improving the energy efficiency include enhancing the resource utilization to reduce resource wastage and applying the renewable energy as the energy supply. This chapter aims to reduce the carbon footprint of the data centers by reducing the usage of brown energy and maximizing the usage of renewable energy. Taking advantage of microservices and renewable energy, we propose a self-adaptive approach for the resource management of interactive workloads and batch workloads. To ensure the QoS of workloads, a brownout-based algorithm for interactive workloads and a deferring algorithm for batch workloads are also proposed. We have implemented the approach in a prototype system and evaluated with real traces. The results illustrate our approach can reduce the brown energy usage by 17% and improve the renewable energy usage by 13%.

7.1 Introduction

APART from self-contained microservices, renewable energy is another solution gaining momentum to address energy consumption concerns (i.e., the carbon footprint) of cloud computing. In response to the climate change concerns and economic stimulus, many research initiatives have been launched to promote renewable energy use to power cloud data centers in recent years [96][71][66]. Many cloud providers also work on this goal by generating their own renewable energy or drawing power from a nearby

This chapter is derived from:
• **Minxian Xu**, Adel Nadjaran Toosi, and Rajkumar Buyya, "A Self-adaptive Approach for Managing Applications and Harnessing Renewable Energy for Sustainable Cloud Computing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2018 (under review).

renewable power plant. For example, in January 2018, AWS achieved 50% renewable energy usage by investing in clean energy activities including a commercial-scale wind farm in North Carolina.[2]

Renewable energy systems are shown to be extremely effective in reducing dependence on finite fossil fuels and decreasing environmental impacts. However, powering data centers entirely or partially with renewable energy sources such as solar or wind is challenging as they are non-dispatchable and not always available due to their fluctuating nature. For example, photovoltaic (PV) solar energy is only available during daytime and the amount of power produced depends on the weather and geographical location of the data center. To be able to offer cloud services under such circumstances, cloud resource management systems need to support methods that allocate resources and schedule applications execution by preferring to finish them during the time when renewable energy is available while at the same time need to make sure that user QoS requirement are honored.

The key **contributions** of the chapter are:

- Providing a perspective model for multi-level adaptive resource scheduling to manage workloads and renewable energy;
- Proposing a self-adaptive approach for interactive workloads and batch workloads to ensure their QoS by considering the predicted renewable energy at Melbourne based on support vector machine technique;
- Implementing a prototype system derived from the perspective model and the proposed approach on a small-scale testbed;
- Evaluating the performance of the self-adaptive approach in the proposed prototype system.

The rest of this chapter is organized as: Section 7.2 discusses the related work for managing energy in the cloud computing environment. Section 7.3 depicts the system model of our proposed approach, followed by modeling and problem statement in Section 7.4. The scheduling algorithm with renewable energy is introduced in Section 7.5.

[2] https://aws.amazon.com/about-aws/sustainability/

Table 7.1: Comparison for related work

Approach	Technique				Energy Efficiency			Workloads Type		Resource Scheduling	
	DVFS	VM Consolidation	Host Scaling	Brownout	Brown Energy	Renewable Energy	Cooling Energy	Single	Mix	Single Layer	Multiple Layer
Beloglazov et al. [33]		√	√		√			√			√
Kim et al. [83]	√				√			√		√	
Liu et al. [94]	√				√		√	√		√	
Teng et al. [140]	√	√	√		√			√			√
Nguyen et al. [117]		√	√		√			√			√
Hasan et al. [74]			√	√	√	√		√			√
Li et al. [90]		√			√		√	√			√
Beloglazov et al. [32]		√	√		√			√			√
Goiri et al. [66]			√		√	√	√		√	√	
Liu et al. [97]			√		√	√	√		√	√	
Our Approach		√	√	√	√	√	√		√		√

Section 7.6 provides the detailed information about the implementation of our prototype system, and Section 7.7 shows the evaluation results of our proposed approach under our prototype system. Finally, conclusions along with the future directions are given in Section 7.8.

7.2 Related Work

A large body of research on the energy efficiency of data centers has been dedicated to the optimization techniques to reduce the energy consumption of servers within a data center using technologies such as dynamic voltage and frequency scaling (DVFS) and VM consolidation [33][83]. Liu et al [94] proposed a heuristic algorithm for big data task scheduling based on thermal-aware and DVFS-enabled techniques to minimize the total energy consumption of data centers. Kim et al. [83] modeled real-time service as real-time VM requests and proposed several DVFS algorithms to reduce energy consumption for the DVFS-enabled cluster. Cheng et al. [45] proposed a heterogeneity-aware task assignment approach to improve the overall energy consumption in a heterogeneous Hadoop cluster without sacrificing job performance. Teng et al. [140] presented a set of heuristic algorithms by taking advantage of DVFS and VM consolidation together for batch-oriented scenarios. Nguyen et al. [117] introduced a virtual machine consolidation algorithm with multiple usage prediction to improve the energy efficiency of cloud data centers. The limitation of DVFS and VM consolidation is that they cannot function well if the whole system is overloaded. Therefore, in this chapter, we take advantage of brownout-based approach to handle the interactive workloads.

Brownout is a self-adaptive approach to manage resources and applications in cloud computing systems. In Chapter 2, we proposed a survey and taxonomy on brownout-based approaches, which summarized that application of brownout in cloud computing systems for different optimization objectives. Tomas et al. [142] applied brownout to address the load balancing issues in clouds. Shahrad et al. [131] proposed a practical pricing model for brownout system and aims to increase the utilization of the cloud infrastructure by incentivizing users to dampen their usage fluctuations. Hasan et al. [74] investigate the green energy and user experience trade-off in interactive cloud applications and propose a controller to provide guarantees of keeping response time within the SLA range in the presence of green energy based on a brownout-enabled architecture.

Due to the complexity of thermal modeling of data center operation, traditional approaches ignored the impacts of resource management techniques on the cooling power system of data centers. Recently, the holistic management of resources in which both computing and cooling energy are considered in the minimization of the overall consumption of energy has gained considerable attention from the community. Li et al [90], for example, provided models capturing thermal features of computer room air conditioning (CRAC) unit of the data center and accordingly propose a VM scheduling algorithm to reduce data center energy consumption while it maintains the SLA violation in an acceptable range. In their work, resource scheduling happens on VM level and the workload type is batch. Al-Qawasmeh et al. [20] presented power and thermal-aware workload allocation in the heterogeneous cloud. They developed optimization techniques to assign the performance states of CPU cores (P-states) at the data center level to optimize the power consumption while ensuring performance constraints. Tang et al. [139] investigated the thermal-aware task scheduling for homogeneous HPC data center, which aims to minimize peak inlet temperature through task assignment, thus reducing the cooling power. However, the virtualized resources are not considered in their model. Compared with these works, we consider multiple layer resource scheduling and mixed types of workloads.

There are many studies in the literature that focused on the optimization of onsite renewable energy use in data centers. Goiri et al [66] presented a prototype of a green data center powered with solar panels, a battery bank, and a grid-tie which they have built as a research platform. They also describe their method, called GreenSwitch, for dynamically scheduling the interactive and batch workload and selecting the source of energy to use. GreenSwitch aims to minimize the overall cost of electricity while respecting the characteristics of the workload and battery lifetime constraints. Similar to our work, they consider batch and interactive workloads in the data center. Their work focuses on the resource scheduling at the application level, while our work is a multi-layer scheduling approach that considers the application, VMs, and hosts. Liu et al. [97] also focus on shifting workloads and matching renewable energy supply and demand in the data center. They schedule non-critical IT workload and allocates IT resources within a data center according to the availability of renewable power supply and the efficiency of the cooling system. They formulate the problem as a constrained convex optimization and aim to minimize the overall cost within the data center. Different from the optimization of overall costs, we aim to optimize the energy perspective. Another difference is that we use two separate algorithms for interactive workloads and batch workloads, while [97] considers the workloads in an integrated manner. The current chapter contributes to the growing body of work.

Table 7.1 shows the comparison among the related work. The most similar works to us are [66] and [96], and we advance them by applying the brownout mechanism and multiple layer scheduling.

7.3 System Model

We propose a system model for adaptive resource scheduling as shown in Figure 7.1 based on the modified perspective model derived from Chapter 2. The main difference lies in the type of workloads and the source of energy supply. We consider both interactive workloads and batch workloads in the application layer and consider green and brown energy together in the energy supply layer.

In the users layers, users submit their service requests to the system. The users can define QoS constraints for the submitted requests, such as budget and deadline. The submitted requests are forwarded to the application layer. From the service providers' viewpoint, these workloads generated by users are processed by the applications hosted in the cloud infrastructure.

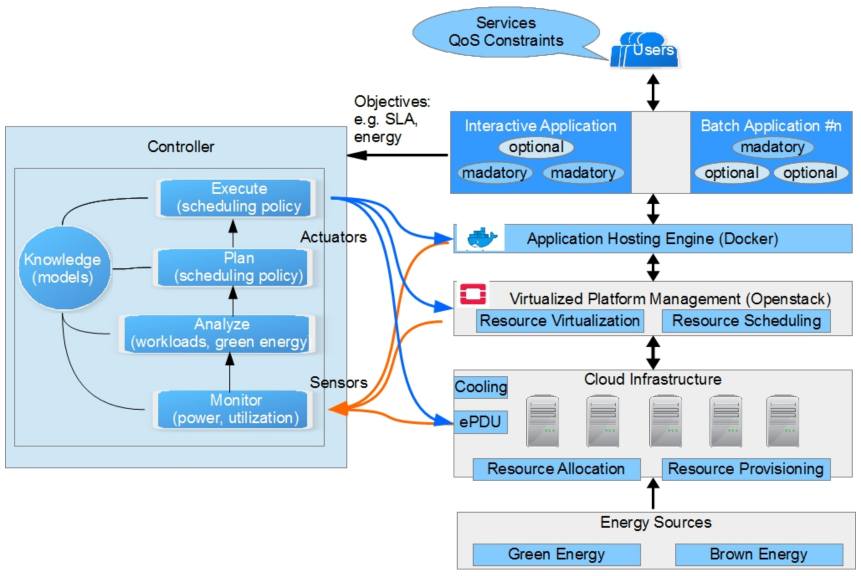


Figure 7.1: Perspective model

We consider two types of applications: the interactive application (such as web application), and batch application. The interactive application should be executed as soon as possible to ensure the QoS. We consider the interactive application to support brownout, thus the microservices of the interactive applications can be identified as optional or mandatory. The optional ones can be deactivated to save resource usage, if deemed necessary. For the batch application, the workloads can be deferred for execution if their deadline is ensured.

Applications provided by service providers to offer services for users are managed by the application hosting engines, such as Docker [11] or Apache Tomcat. Applications can be deployed on either virtualized platform (virtual resources) or cloud infrastructure (physical resources). The host application engine can be container-based management platforms, e.g. Docker Swarm [17], Kubernetes [16] or Mesos [13], which provide management of container-based applications. The virtualized platform manages the virtualized resources, for example, the Virtual Machines managed by VMware [18]. As for the resource allocation and provisioning in cloud infrastructure, they can be managed by infrastructure management platform, e.g. OpenStack [15].

The bottom layer is the energy supply layer, which provides the mixed of energy sources for powering the system. The brown energy comes from a coal-based thermal power plant, which has high carbon footprint. The green energy comes from the renewable energy, such as solar power.

To support the resource provision, monitor and allocation in the system, a controller is required based on MAPE-K architecture model and fits into the feedback loop of the MAPE-K process, which has modules including *Monitor*, *Analyze*, *Plan* and *Execute* to achieve the adaptation process in cloud computing system. Sensors and Actuators are used to establish interactions with the system. Sensors gather the information from different levels in the system, including application hosting engine, virtualized platform, cloud infrastructure, and energy usage. The Sensors can be the devices attached to hardware, e.g. power meter. The collected information is provided to the Monitor module.

The Analyze module analyzes the received information from the Monitor module, and the Plan module makes decisions for applying scheduling policies, in which the scheduling policies are implemented. According to the decisions, the Execute module schedules resources via actuators on the application hosting engine and the virtualized platform to enable/disable optional microservices in interactive applications or defer the workloads of batch applications to be supplied by renewable energy. These operations can be fulfilled via the APIs provided by the application hosting engine or the virtualized platform.

The Knowledge pool in MAPE-K model is applied to store the predefined objectives (energy efficiency or SLA constraints) and trade-offs (e.g. trade-offs between energy and SLA). The rules in Knowledge pool, such as SLA rules, can be updated according to resource scheduling algorithms. The Knowledge pool also contains models like predicting the supplied amount of renewable energy, which can be used by scheduling algorithms.
.

In the following sections, we will introduce our proposed approach and the prototype system that is derived from this perspective model.

7.4 Problem Modeling

In this section, we will discuss the modeling and the optimization problem. Table 7.2 shows the symbols and their meanings utilized throughout this chapter.

Table 7.2: Symbols and definitions

Symbol	Definition
i	Server (host) i
P_i^s	Power consumption of server i
P_i^{idle}	Power when i is idle
$P_i^{dynamic}$	Power when i is fully loaded
w_i	The number of VMs deployed on host i
$U_{i,j}^{vm}$	The utilization of the jth VM on host i
t	The time interval
T	The scheduling period
θ_t	The brownout dimmer value at time interval t
A_j	The number of microservices on VM j
U_j^{ms}	The utilization of microservice ms
CoP	The function to calculate the cooling efficiency of cold air
T_{sup}	Cooling air supply temperature
P_i^c	Cooling power for host i
P_i	Total power of host i
P_t	Total power of data center at time interval t
$d(t)$	Total workloads at time interval t
$a_m(t)$	The interactive workloads at time interval t with size m
$b_n(t)$	The batch workloads at time interval t with size n
D	The maximum resource capability in system
S_n	The start time of batch workload $b_n(t)$
E_n	The execution time of batch workload $b_n(t)$
D_n	The deadline of batch workload $b_n(t)$
sla	SLA violation ratio in the system
num_f	The number of failed requests
num_a	All the number of requests coming into system
R_t	Available renewable energy at time interval t
$d(t)^+$	Power resulted from workloads on server
$c(d(t))$	Power resulted from cooling part reduction
α	Allowed SLA violation ratio
$avg_a(t)$	Average response time interactive workloads at time interval t
β	Allowed average response time
U_i^t	Host utilization of host i at time interval t
TU_{up}	Threshold to determine overloaded hosts
TU_{low}	Threshold to determine underutilized hosts
t_r^s	The start time of available renewable energy
t_r^e	The end time of available renewable energy
n_o^t	The number of overloaded hosts at time interval t
n	The total number of hosts in the system
ϵ	The percentage of utilization from batch workloads
S_i	The set of deactivated microservices on host i
$U(S_i)$	The utilization sum of deactivated microservices
$d(t)^{'}$	Predicted total workloads at time interval t
$a_m(t)^{'}$	Predicted interactive workloads at time interval t with size m
$b_n(t)^{'}$	Predicted batch workloads at time interval t with size n
T_j^D	Deferred time of batch workload
td	Start time of batch workload after deferred
$S_j^{'}$	Start time after deferred
$R_{td}^{'}$	Predicted renewable energy at td
$P_{td}^{'}$	Predicted power consumption at td
n_a	The number of active hosts
num_{thr}	Number of requests when host is overloaded
$n^{'}$	The difference between required host and active hosts

7.4.1 Power Consumption

Server Power Consumption

The server power model is derived from [155], which is based on the average CPU utilization. As we consider multiple-layer scheduling, the utilization of hosts, VM and microservices are modeled:

$$P_i^s = \begin{cases} P_i^{idle} + \theta_t \sum_{j=1}^{w_i} U_{i,j}^{vm} \times P_i^{dynamic} & , w_i > 0 \\ 0 & , w_i = 0 \end{cases} \tag{7.1}$$

where P_i^s is the power consumption of the host i in the data center, which is composed of two parts: the idle power P_i^{idle} and dynamic power $P_i^{dynamic}$. The dynamic power part is related to the VM utilization on the host. If there is no VM running on the host, it means the host can be switched into the low power mode and consume low energy. The w_i represents the number of VMs deployed on host i. $U_{i,j}^{vm}$ represents the utilization of jth VM on host i. The θ_t is the dimmer value of brownout at time interval t.

The utilization of VM is the sum of microservices utilization running on the VM, which is modeled as:

$$U_{i,j}^{vm} = \sum_{k=1}^{A_j} U_{j,k}^{ms} \tag{7.2}$$

where the ms is the id of microservice and A_j is the number of microservices. Since CPU computation is main power consumption component of servers, in our server model, we mainly focus on the power draw by the CPU utilization.

Cooling Power Consumption

For cooling power consumption, we use the model from HP lab data center [112] as follows:

$$CoP(T_{sup}) = 0.0068T_{sup}^2 + 0.0008T_{sup} + 0.458 \tag{7.3}$$

The CoP in Equation 7.3 is a function to estimate the cooling efficiency of cold air supply temperature T_{sup} provided by cooling equipment, which is related to the target temperature that room is aimed to be maintained.

We consider the data center thermal control is managed by Computer Room Air Condition (CRAC) system. The system contains multiple CRAC units, which transfers cold air to the hosts to reduce hotspots. Based on server power consumption and cooling efficiency, we can calculate the power consumed P_i^c by cooling equipment for host i as:

$$P_i^c = \frac{P_i}{CoP(T_{sup})} \tag{7.4}$$

The total power draw by the server part and the cooling part can be represented as:

$$P_i = P_i^s + P_i^c \tag{7.5}$$

The total power of the data center with n servers:

$$P_t = \sum_{i=1}^{n} P_i \tag{7.6}$$

7.4.2 Workloads Model

In this work, we consider two types of workloads: (1) interactive workloads and (2) batch workloads. The interactive workloads are response time sensitive, thus these workloads should be executed immediately with the response time specified in the SLA, while the batch workloads can be deferred for execution as long as the deadline is satisfied. Based on the different characteristics of these workloads, the interactive workload at time t is denoted as $a_m(t)$, and the batch workload is presented as $b_n(t)$, with start time S_n, execution time E_n, and deadline D_n. Therefore, the total workload at t is:

$$d(t) = \sum_{m} a_m(t) + \sum_{n} b_n(t) \tag{7.7}$$

The value of $d(t)$ should be $0 \leq d(t) \leq D$, in which D is the maximum resource capability of the system.

We use *sla* to denote the SLA violation that the system allows for interactive workloads, which represents the ratio of the requests fail to execute within predefined SLA, which is modeled as:

$$sla = \frac{num_f}{num_a} \tag{7.8}$$

The num_a is the number of all the requests coming into the system, and num_f is the number of failed requests. To ensure the QoS, we have the constraint on all requests as $sla \leq \alpha$. For the interactive requests the average response time $avg_b(t)$ should be below the threshold defined by service provider $avg_a(t) \leq \beta$.

7.4.3 Optimization Objectives

We assume the scheduling period as T, and time interval we schedule resources is denoted as t. We assume the available renewable energy at time t is R_t. As the server power and cooling power is related to workloads, we use $d(t)^+$ to denote the power consumption resulted from the workload execution on servers, and $c(d(t))$ represents the cooling power resulted the from workload, thus $P_t = d(t)^+ + c(d(t))$, our optimization objective is modeled as:

$$\begin{aligned} min \sum_t (max(P_t - R_t, 0)) \\ s.t. 0 \leq d(t) \leq D, \ \forall t \\ 0 \leq \theta_t \leq 1, \ \forall t \\ sla \leq \alpha, \forall t \\ avg_b(t) \leq \beta, \forall t \end{aligned} \tag{7.9}$$

7.5 Scheduling with renewable energy

7.5.1 Green-aware Scheduling Algorithm

To schedule the interactive and batch workloads in an energy efficient manner by considering renewable energy, we propose a Green-aware scheduling algorithm, which is shown in Algorithm 7.1. During the observation period T, at each time interval t, the algorithm will firstly identify the number of overloaded hosts (line 2). If the overloading situation exists, the algorithm will manage the interactive workloads and batch workloads with different algorithms: brownout algorithm for interactive workloads (Algo-

Algorithm 7.1: Green-aware scheduling algorithm

```
Input  : host utilization $U_i^t$, utilization thresholds $TU_{up}, TU_{low}$
Output: brown energy usage $\sum_t(max(P_t - R_t, 0))$
1  $n_o^t = \sum_k (U_i^t > TU_{up})$
2  for $t \leftarrow 0$ to $T$ do
3      if $n_o^t > 0$ then
4          $S_i \leftarrow$ brownout algorithm for interactive workloads
5          $T_j^D \leftarrow$ deferring algorithm for batch workloads
6          min $\leftarrow \sum_t(max(P_t - R_t, 0))$
7      else if $U_{avg}^t < TU_{low}$ then
8          VM consolidation algorithm
9          host scaling algorithm
10         min $\leftarrow n_a$
11     else
12         process iteractive workloads in normal mode
13         porcess batch workloads in normal mode
14     end
15 end
```

rithm 7.2) and deferring algorithm for batch workloads (Algorithm 7.3) to minimize brown energy usage (lines 4-6). If the system is not overloaded and average utilization is below the underutilized threshold (line 7), the algorithm will apply VM consolidation algorithm to pack VMs on fewer servers, thus the idle servers will be switched into the low power mode to save energy (lines 8-10). If the system is running at the normal status, then the workloads will be executed in the normal fashion.

7.5.2 Brownout Algorithm for Interactive Workloads

The pseudocode of the brownout algorithm for interactive workloads is shown in Algorithm 7.2. The algorithm schedules resources differently according to whether the renewable is available or not. 1) During the time when renewable energy is not available (line 2), the brownout is triggered, and the dimmer value is generated. The dimmer value θ_t is computed based on the severity of overloads in the system (line 3). With the dimmer value, the expected utilization reduction U_i^r on host i is computed (line 4). Then the algorithm selects a set of microservices S_i to deactivate, thus the utilization is reduced. The difference between the expected utilization reduction U_i^r and the sum of utilization of selected microservices $U(S_i)$ is minimized (lines 6-8). To minimize the difference, the microservice selection process is based on the LUCF algorithm in Chapter 3 , which sorts the microservices according to their utilization in a list, and finds the sublist which has the

Algorithm 7.2: Brownout Algorithm for Interactive Workloads

```
Input  : time interval $t$, the number of overloaded hosts $n_o^t$
Output: deactivated microservice $S_i$
for host i in the host list do
    if $t < t_r^s \,||\, t > t_r^e$ then
        $\theta_t = \sqrt{\frac{n_o^t}{n}}$
        $U_i^r = \theta_t \times U_i^t$
        if $U_i^t > TU_{up}$ then
            finding deactivated microservices $S_i$ on host $i$
            min $\leftarrow |U_i^r - U(S_i)|$
            deacivate the microservices
        end
    else
        if $R_t < P_t$ then
            $\theta_t = \frac{1}{1-\epsilon} \times \sqrt{\frac{R_t}{P_t}}$
            $U_i^r = \theta_t \times U_i^t$
            finding deactivated microservices $S_i$ on host $i$
            min $\leftarrow |U_i^r - U(S_i)|$
            deacivate the microservices
        end
    end
end
```

utilization that is closest to U_i^r. 2) When the renewable energy is available but less than the total required energy, the brownout is also triggered (line 11). The dimmer value is calculated based on renewable energy and required energy as noted in line 12. Then the rest steps are the same as in the first part of Algorithm 7.2, which finds the microservices and deactivate them. 3) When sufficient renewable energy is available, brownout will not be triggered.

7.5.3 Deferring Algorithm for Batch Workloads

Algorithm 7.3 shows pseudocode for processing the batch workloads. The batch workloads are executed when their start time S_j is coming (line 1). The workloads are processed based on the time period that the workloads are in. For the workloads which have the start time before the renewable available start time t_r^s, the objective is to defer their execution to the time when the renewable energy is available while ensuring their deadlines (lines 2-12). 1). If the deadline is before t_r^s, it means the workload cannot be deferred to be processed by renewable energy, so the workload can be executed at t (lines 3-4). If the workload can be deferred, the algorithm defers its time with T_j^D, then the algorithm

Algorithm 7.3: Deferring Algorithm for Batch Workloads

```
Input  : batch workload $b_n(t)$ with start time $S_j$, execution time $E_n$, and deadline $D_n$
Output: deferred time $T_j^D$
for $t = S_n$ in $b_n(t)$ do
    if $t > 0$ & $t < t_r^s$ then
        if $D_j < t_r^s$ then
            execute $b_j(t)$
        else
            defer $T_j^D$ time for execution
            $td = t + T_j^D, \forall\, t_d \leq D_j - E_j, td > t_r^s$
            $d(td)' = \sum_m a_m(td)' + \sum_n b_n(td)'$
            $R'_{td} > P'_{td}$
            $S'_j = td$
            update $P'_{td}$
        end
    else if $t_r^s \leq t \& t \leq t_r^e$ then
        if $R_t > P_t$ then
            execute $b_n(t)$
        else
            defer $T_j^D$ time for execution
            $td = t + T_j^D, \forall\, t_d \leq D_j - E_j$
            $d(td)' = \sum_m a_m(td)' + \sum_n b_n(td)'$
            $R'_{td} > P'_{td}$
            $S'_j = td$
            update $P'_{td}$
        end
    else
        execute $b_n(t)$
    end
end
```

updates the workloads at time td, which equals to $t + T_j^D$. The deferred time T_j^D should satisfy the constraint, e.g. not failing the deadline, the renewable energy is enough at td, and should not be deferred to after t_r^e. If the constraints are satisfied, the algorithm updates the predicted power consumption at td. 2). When the start time of the workload is during the time when renewable energy is available if the renewable is sufficient, the workload is executed; otherwise, the workload will be deferred. Similar to the first part of Algorithm 7.3, the deferred time td also needs to satisfy the constraints in Equation 7.9 but removing the constraint that workload is executed before t_r^e. 3) When the time is after t_r^e, it means the renewable energy is not available any more, therefore, the workloads are executed as soon as possible to comply with the deadlines.

7.5.4 Host Scaling

Algorithm 7.4: Hosts scaling algorithm

```
Input  : number of hosts $n$ in data center, number of active hosts $n_a$, number of requests when host
         is overloaded $num_{thr}$, predicted number of requests $\hat{num}(t)$ at time $t$.
Output: number of active hosts $n_a$
$n_a \leftarrow \lceil \hat{num}(t) \div num_{thr} \rceil$
$n' \leftarrow n_a - n$
if $n' > 0$ then
    Add $n' hosts$
    while $P_t \leq R_t$ do
        Add another host
        update $P_t$
    end
else if $n' < 0$ then
    Remove $|n'|$ hosts
else
    no host scaling
end
return $n_a$
```

We use a modified host scaling algorithm from [143] by considering renewable energy as shown in Algorithm 7.4. With profiling data, we configure the threshold of requests that leads to overloads, in which the average response time violates the predefined constraints. The algorithm calculates the difference $n^{'}$ between the number of required servers and actual servers. 1). When more servers are needed, then it adds $n^{'}$ servers into the system (lines 3-4). If the renewable energy is still enough, then it tries to scale more servers into the system to improve the QoS (lines 5-8). 2). If servers are already enough, then remove $|n^{'}|$ servers from system to reduce energy. 3). If $n^{'}$ is 0, then it means no host scaling is required.

7.5.5 Renewable Energy Prediction

In this work, we focus on the solar energy as it is one of the most common sources of renewable energy. We use Support Vector Machine (SVM) to predict the solar irradiation or PV power output for the availability of renewable energy. SVM is a machine learning approach and has been applied to data analysis successfully. In the studies related to solar irradiation prediction, SVM has been used to forecast and train solar radiance model [27][30]. Besides, the amount of weather data that we obtained is not as huge as the dataset like human's faces. Thus, instead of training with more advanced machine learning techniques, we choose SVM.

In this chapter, we gather the historical data of Melbourne city, Australia and use SVM to train a prediction model. We use the daily and monthly mean attributes, including maximum temperature, minimum temperature, maximum humidity, minimum humidity, sunshine duration and solar radiation from Bureau of Meteorology of Australia [3]. The solar radiation is the output, and other attributes are considered as input. SVM prediction approach has two phases: the training phase and testing phase. Three years data are used for the training phase, and one year is used for the testing phase. Once the process is finished, the test data and prediction results are compared to calculate the error rate. We use SVM R toolbox for our purpose.

The obtained results are shown in Figure 7.2. We use two SVM models with different kernels. It shows the svm-2 achieves better performance than svm-1. The root mean square error (RMSE) and the correlation coefficient are equals to 1.687 and 0.97. The selected parameters for svm-2 are regularization parameter $C = 4$ and Kernel bandwidth $\epsilon = 0$. In the experiments section, we use svm-2 model to predict solar irradiation with the aforementioned settings.

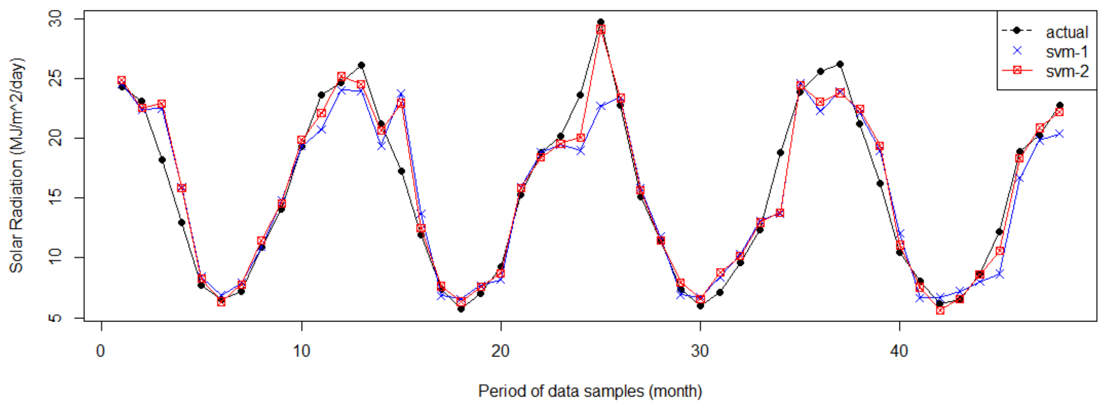


Figure 7.2: Melbourne solar radiation

[3] http://www.bom.gov.au/climate/data-services/solar-information.shtml

7.6 Prototype System Implementation

We configure our testbed in Chapter 6 to develop a prototype system to evaluate the proposed approach. Figure 7.3 shows the implemented architecture of our prototype system. Cloud resource management platform and microservices management platform have been developed and widely used for years, thus, in this work, we design and implement our prototype based on these mature platforms.

Cloud IaaS resource management platform, OpenStack, is responsible for managing cloud resources, including CPU, memory, and bandwidth. The monitored data of resources is collected by status collector and can be used for resource provisioning and optimization. Microservice management platform, Docker Swarm, is responsible for managing service images, monitoring service resource utilization and managing the service life cycles. Other Docker APIs can also be used to run operations on services.

Based on the two management platforms for cloud resources and services, SA (Self-Adaptive) controller is designed to manage and monitor both of them to achieve the multiple level resource scheduling. When requests are submitted to the system, like interactive workloads or batch workloads, the resource allocator in SA controller manages cloud resource management platform and service management platform simultaneously to accept and process requests by providing the requested amount of resources. Apart from allocating resources to requests, the resource allocator can also optimize resource utilization. For instance, brownout can be triggered to deactivate optional microservices to reduce resource utilization. The service provider can also configure the available resource scheduling policies for the energy efficiency purpose.

To provision and optimize the resources by means of resource allocator, the resource monitor needs to collect the resource usage at different levels, including services utilization, VMs utilization, and hosts utilization. To minimize the overheads of frequently monitored data collection, the collection time intervals should be well configured by the service provider. For instance, the brownout mechanism can be checked every five minutes as the brownout costs are not high, while the VM migration and host scaling operations can be executed with longer time intervals, e.g. one hour.

In the following subsections, we introduce the implementation of our prototype system in details.

7.6.1 Implementation

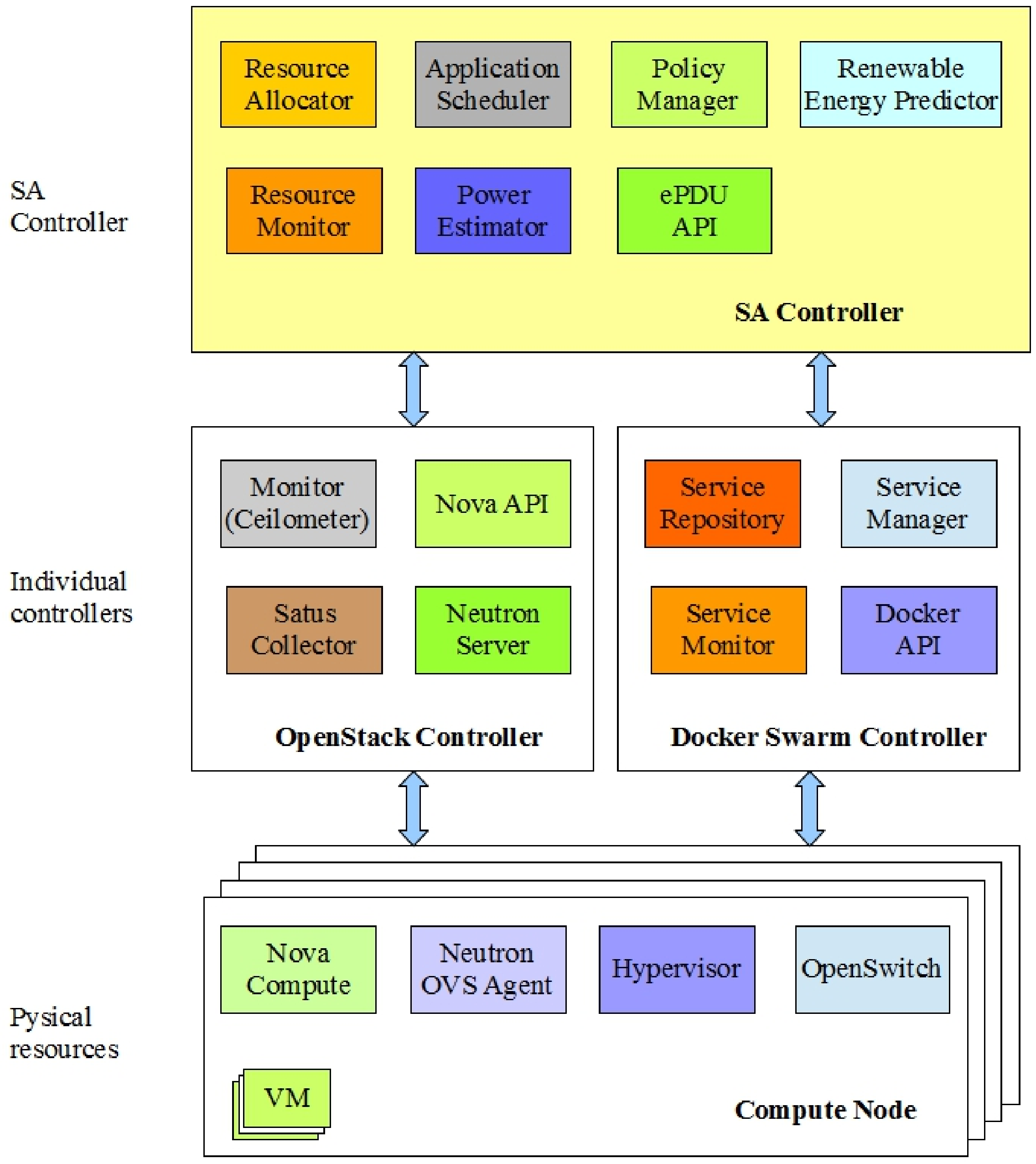


Figure 7.3: System architecture and underlying software components of prototype system

To implement our prototype system, we take advantage of the OpenStack cloud resource management platform and Docker Swarm service management platform. The system is implemented with Java, OpenStack, Docker Swarm, Ansible, Eaton Power Distribution Units (ePDU) API. Our prototype system uses these open source tools to provide a self-adaptive approach to optimize, manage and provision resources for different types of workloads.

OpenStack platform is used to manage hosts and VMs. The hosts in OpenStack are called compute nodes and are running with Nova Compute Node component to connect the hypervisor and OpenStack controller. VMs are managed by Nova API to create, migrate and remove VM instances. The Neutron OVS Agent and OpenVSwitch are providing services related to the network.

Docker Swarm Platform manages the service provided by service providers. The images of services are stored in the service repository component, which can fetch the images from remote to local. The services are managed by the service manager via Docker APIs, including creation and deletion. The status of services are monitored by a service monitor where we monitor service utilization and liveness.

Our prototype system is based on these services to manage the resources and services to handle the requests. Below, we introduce the details of the components in our prototype.

Resource Allocator: It interacts with OpenStack controller via OpenStack APIs and Docker Swarm Controller via Docker APIs. It manages the physical resources on compute nodes, and these physical resources can be used for creating and deploying VMs on the nodes. Resource Manager knows the amount of resource that is used or remaining on each compute node, like the the number of cores, memory, and storage. When creating a VM instance, it can also specify the instance capacity (CPU, memory, operation system and etc.) as well as other information related to VMs, such as location, images of VMs and IP address. The virtual network in a compute node is also managed by Resource Manager that uses the Neutron component, which is deployed on each compute node.

Resource Monitor: It is used to monitor the running status of the whole system from different levels, including hosts, VMs and services. We use OpenStack Ceilometer and Gnocchi components to measure the data at the host and VM level. Ceilometer is responsible for monitoring the utilization of resources of VMs and then sends the collected data to Gnocchi to aggregate the data for all the hosts. We use Docker APIs to collect the resource utilization of services deployed on VMs. Apart from monitoring the resource utilization, we also use ePDU APIs to monitor the power consumption of hosts. With these monitored data, other components, like Power Estimator and Policy Manager can use these data to make decisions, which will be introduced later.

Application Scheduler: We design our main controls in the application scheduler component. When requests are submitted by users, the application scheduler decides which requests in the batch workloads should be deferred, which microservice should be temporarily deactivated by brownout mechanism, which VM should be migrated to another host and which host should be switched to the low power mode. With the retrieved data from the Resource Monitor component, these decisions are made with the policies in the Policy Manager. After the decisions are made, Resource Provisioner exploits Resource Manager to allocate the resources to VMs, services, and requests.

Power Consumption Estimator: To achieve our objective of managing energy and support our scheduling policies, we have a power consumption estimator to predict the power consumption at a specific time period. For example, for the batch workloads, we proposed a deferring algorithm, thus we need to estimate the power consumption at the deferred time period to calibrate our algorithm. We use the workloads model shown in Equation 7.7 to estimate the workloads and then convert it to the total energy consumption based on the model in [96].

Policy Manager: It contains the implemented scheduling policies in our prototype, including Algorithms 7.1 to 7.4.The Policy Manager component uses the retrieved data from Resource Monitor, and makes decisions based on system status. For example, a VM is migrated from an underutilized host to other hosts, thus the idle host can be switched to the low power mode to save power consumption; when the renewable energy is not sufficient and the system is overloaded, to ensure the QoS of service, brownout can be triggered to relieve the overloaded situation. The customized workloads processing policy, VM migration policy and host scaling policy can also be implemented for the policy manager.

ePDU API: Eaton Power Distribution Units (ePDU) [4] is an effective power management and monitoring device. It has outlets that electric devices can be connected to it. It also provides the features to read the power consumption of hosts as well as turn on/off the outlets remotely. We implemented Python scripts based on ePDU APIs to read the

[4]https://powerquality.eaton.com/ePDUG3/

Table 7.3: Machines specification

Machine	CPU	Cores	Memory	Storage	Idle Power	Full Power
3 × IBM X3500 M4	2 GHz	12	64 GB	2.9 TB	153 Watts	230 Watts
4 × IBM X3200 M3	2.8 GHz	4	16 GB	199 GB	60 Watts	150 Watts
2 × Dell OptiPlex 990	3.4 GHz	4	8 GB	399 GB	26 Watts	106 Watts

power data at per second rate to support part of the functions in Resource Monitor. Our scripts can also operate the hosts remotely by turning on/off the power supply to hosts to support the decision in Policy Manager. For example, a host needs to be scaled out if the whole system is underutilized; or hosts should be scaled in to support more requests.

Renewable Energy Predictor: For supporting our renewable energy experiments, we implement a renewable energy predictor that predicts the renewable energy at Melbourne city based on the historical data. Our current renewable energy predictor is based on the support vector machine. As introduced in Section 7.5.5, our SVM models show that it can achieve a high accuracy. The data based on this component can also be incorporated into the scheduling policy. In our current implementation, we consider five attributes that introduced in Section 7.5.5 as inputs, more attributes can be considered as a customized renewable energy predictor.

7.7 Performance Evaluation

7.7.1 Environmental Settings

Hardware: We utilize a micro data center of Melbourne CLOUDS lab as our testbed. Our data center is consist of 9 heterogeneous servers. Table 7.3 shows the capacity specification of the servers and their energy consumption information. To monitor the power consumption of individual machines, we use two Eaton EMAB03 vertical managed enclosure Power Distribution Units (ePDUs) and all the servers are connected to them. Apart from the power monitor, Eaton ePDUs also enable us to switch on/off power outlets connected with individual server remotely through the network. Figure 7.4 shows the servers and ePDUs of our testbed. The total maximum power of the IT equipment in our system is 1.27 KWh for 8 hosts (one IBM X3500 M4 machine is regarded as the OpenStack control node and is not considered).

We assume our system is equipped with 1.63 KWh PV panel, which has 30% more power than the hosts, as the cooling part normally consumes about 20% to 30% of server energy if the target temperature is 25 degree [96]. We consider to control the data center temperature as 25 degree, according to Equation 7.3, $T_{sup} = 25$, then we get $CoP(T_{sup}) = 0.211$. In the following experiments, we use this value to compute the power from the cooling equipment, e.g. if the hosts consume 10 kWh, then the cooling part is 2.11 kWh.

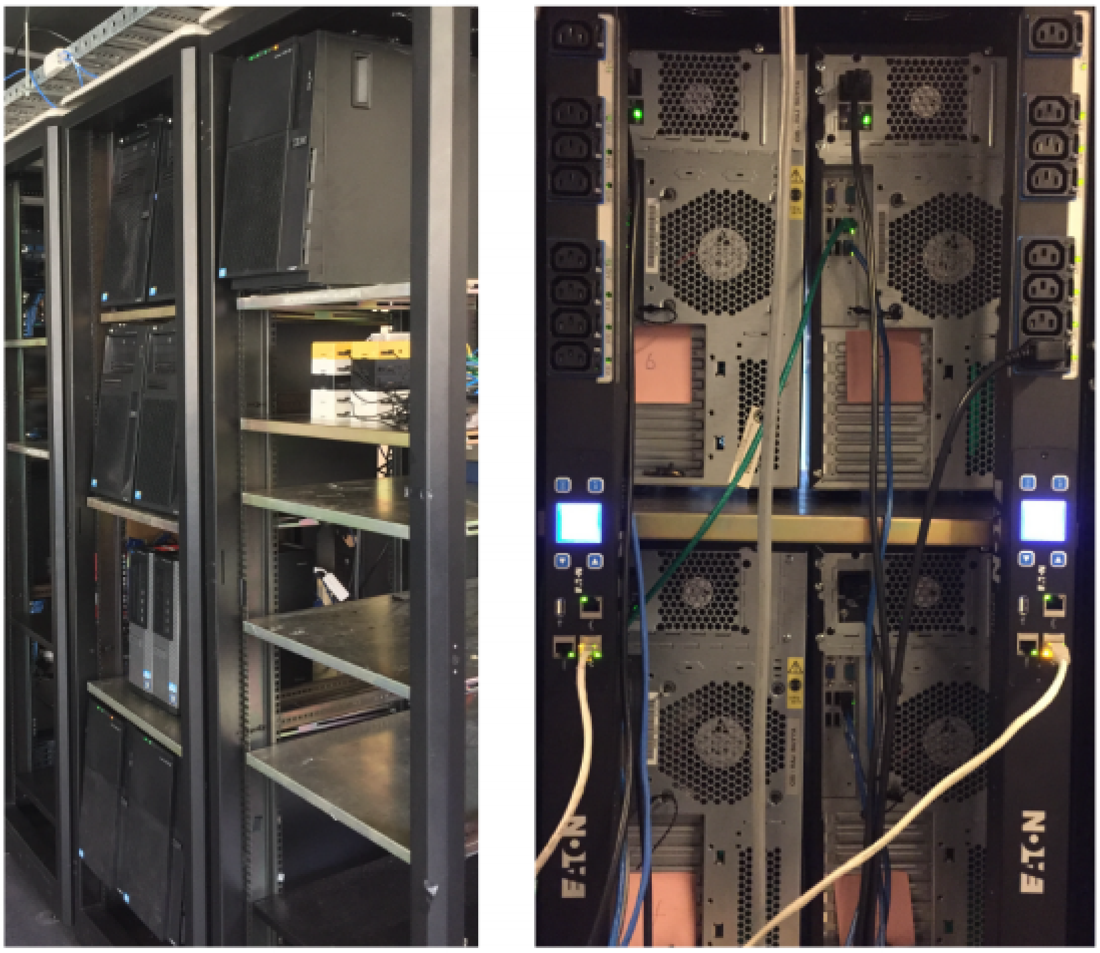


Figure 7.4: Testbed

Software: The operating systems of the servers are CentOS Linux Distribution. We use OpenStack [15] to support our cloud platform and manage the VMs. One of our most powerful machines is selected as our controller node, and other nodes are acting in the same role. In VM instances, we deploy Docker [55] containers to provide services in the form of microservices and use Docker Swarm to manage the containers cluster. Some other required software, like Java, Ansible are also installed in the VMs.

7.7.2 Workload

To make the experiments as realistic as possible, we use real traces from Wikipedia and Facebook. For the interactive workloads, we use the real trace from Wikipedia requests on 2007 October 17 to replay the workload of Wikipedia users. The trace includes data on requests time stamp and their accessed pages. We filter the requests based on per second rate and generate the requests rate. The original request rate is around 1,500- 3,000 per second.

For the batch workloads, we use the traces collected in October 2009 by Facebook for applications that are executed under Hadoop environment[5]. Referring to [66], we configure the map phase of each job takes 25-13000 seconds, and the reduce phase takes 15-2600 seconds. The deadline for processing jobs is generated based on uniform distribution with $\mu = 6$ hours and $\sigma = 1$ hour in $N(\mu, \sigma^2)$. We also assume the workloads consume the maximum of cluster utilization as 27% as same as in [66].

7.7.3 Application

We use the Weave Shop[6] web application that implemented with containers as the application in our scenario. The Weave Shop is a shopping system for selling socks online and has multiple microservices, including user microservice to handle user login, user database microservice for user information storage, payment microservice to process transactions, font-end microservice to show the user interface, catalog microservice for managing item for sale and etc. As these microservices are implemented independently, they can be deployed and controlled without impacting other microservices. The application is deployed by a configuration file, and part of the microservices are configured as optional, e.g. recommendation engine.

[5]https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository
[6]See https://github.com/microservices-demo/microservices-demo for more details.

7.7.4 Results

To evaluate the benefits of our proposed approach for renewable energy usage, we perform the comparison of our proposed approach (SA) and a baseline algorithm (HS), which applies VM consolidation [35] and host scaling [143] that dynamically adds/removes hosts in system, while the green-aware scheduling algorithm in our proposed approach is not applied.

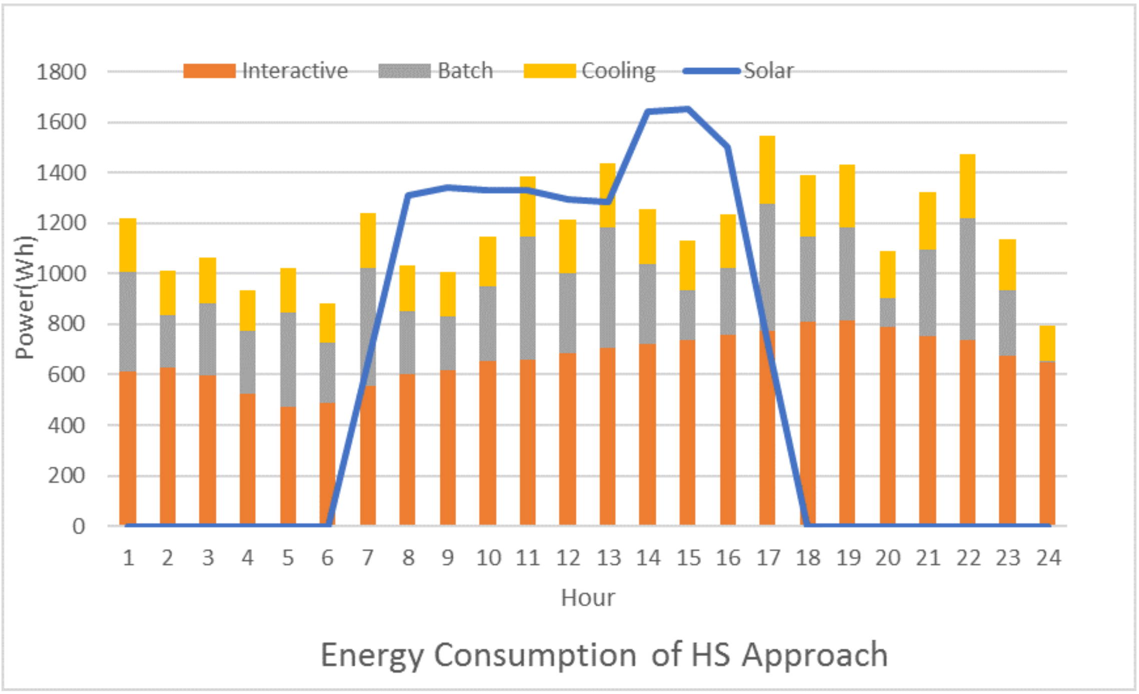


Figure 7.5: Results of baseline

Figure 7.5 shows the baseline energy consumption of interactive workloads, batch workloads and cooling during the observed time period (one day). The blue line shows the predicted solar energy based on our SVM model. The system is consuming brown energy at *night time* from 0:00 to 6:00 and 18:00 to 24:00. The solar energy is available at *day time* during hours 7:00 to 17:00. Even with taking advantage of VM consolidation and host scaling, the solar energy consumption of the system is not fully utilized. For example, at hour 9:00, the total energy consumption of system is about 1000 Watts, while the available solar energy is more than 1300 Watts.

Figure 7.6 demonstrates the energy consumption of our proposed approach by using Algorithms 7.1 to 7.4. We can observe that the power consumption of the batch work-

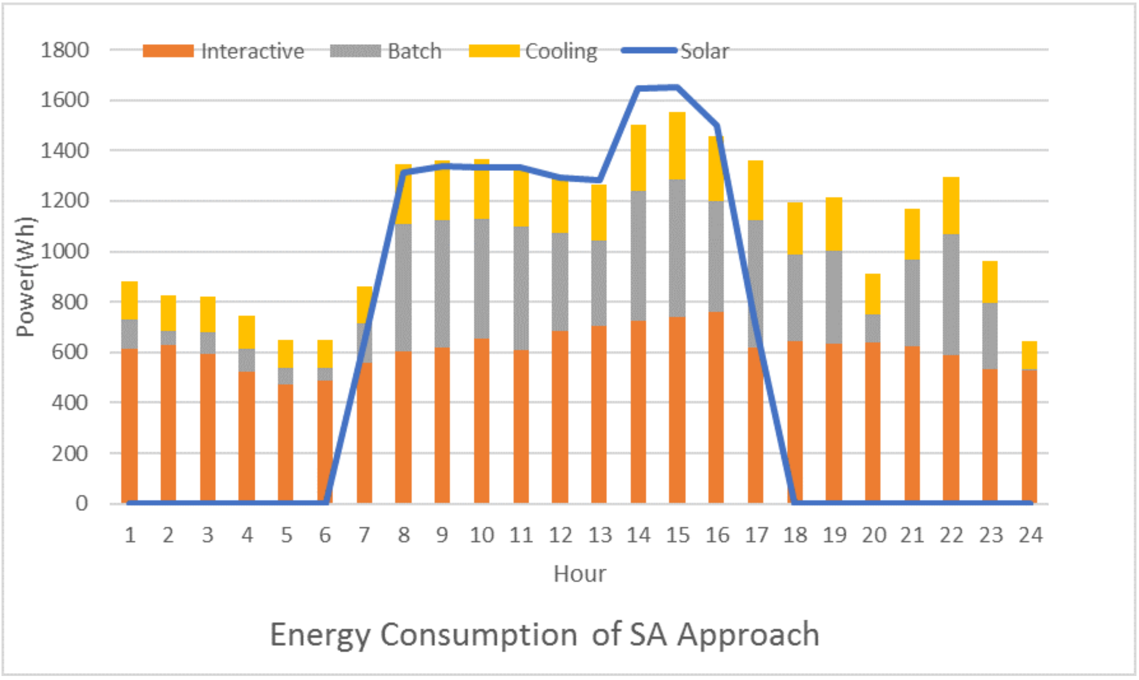


Figure 7.6: Results of proposed approach

loads during 0:00 to 6:00 has been reduced, which results from the deferring operations: batch workloads are deferred to the time when solar energy is available, e.g. hour 8:00. If the deadline of these workloads is before the end time the solar energy. Some batch workloads are still executed during hours from 0:00 to 6:00 due to the deadline constraints, which cannot be deferred to the time when renewable energy is available. Thus, we can find that the brown energy usage during 0:00 to 6:00 has been reduced compared in Figure 7.5. For example, at hour 1:00, the total power is reduced from 1221 to 882 Watts.

During the time when solar energy is available, our proposed approach has improved the usage of renewable energy, in which the energy consumption follows the line the of predicted renewable energy. For instance, at hour 8:00, the usage of solar energy is increased from 1238 Wh to 1345 Wh. During the hours 14:00 and 15:00, the system cannot utilize the full renewable energy because of the maximum IT equipment power.

We can also note that the power consumption during the time when brown energy is the only source of power supply, the energy is also reduced, which exploits the brownout mechanism to reduce the energy consumption. For instance, the power at hour 18:00 is decreased from 1391 Watts to 1195 Watts.

Combing the results in Figures 7.5 and 7.6, we conclude that our proposed approach can improve the usage of renewable energy and reduce the usage of brown energy.

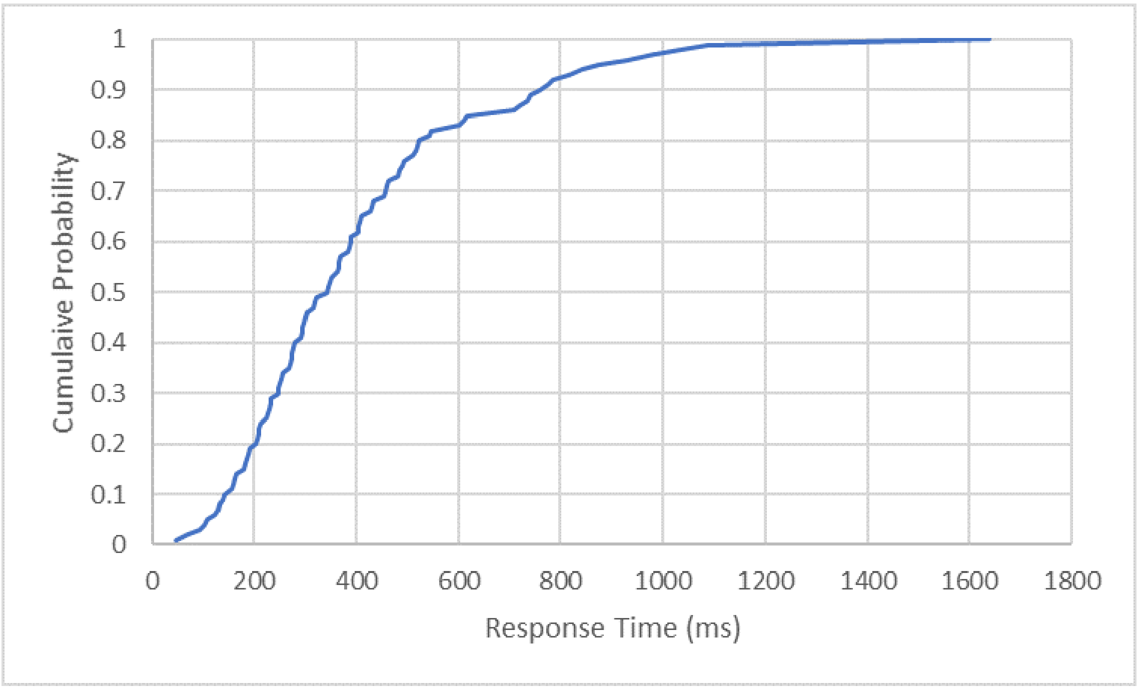


Figure 7.7: Response time of proposed approach

The average response time and CDF response time for interactive workloads in our proposed approach are demonstrated in Figure 7.7. The overall measured response time is 403 ms. The results also show that 95% of requests are responded in 900 ms, and 99% of requests are responded within 1 second. It shows that our proposed approach reduces brown energy usage while ensuring the QoS. The reason is that the brownout approach can relieve the overloaded situation, thus ensuring the response time.

To illustrate the reason of power reduction by our proposed approach, Figure 7.8 compares the active hosts during the observed time period between the baseline and our proposed approach, as switching the idle hosts into the low power mode is the most effective way to save power. The results demonstrate that our proposed approach uses fewer hosts during the time period when renewable energy is not sufficient, e.g. hours from 0:00 to 7:00 and 17:00 to 24:00, while when the renewable energy is available, more hosts are scaled in to utilize more renewable energy, such as the time from 7:00 to 12:00. In this way, the power of all the active hosts is reduced.

Figure 7.9 demonstrates the comparison of brown and renewable energy usage. Dur-

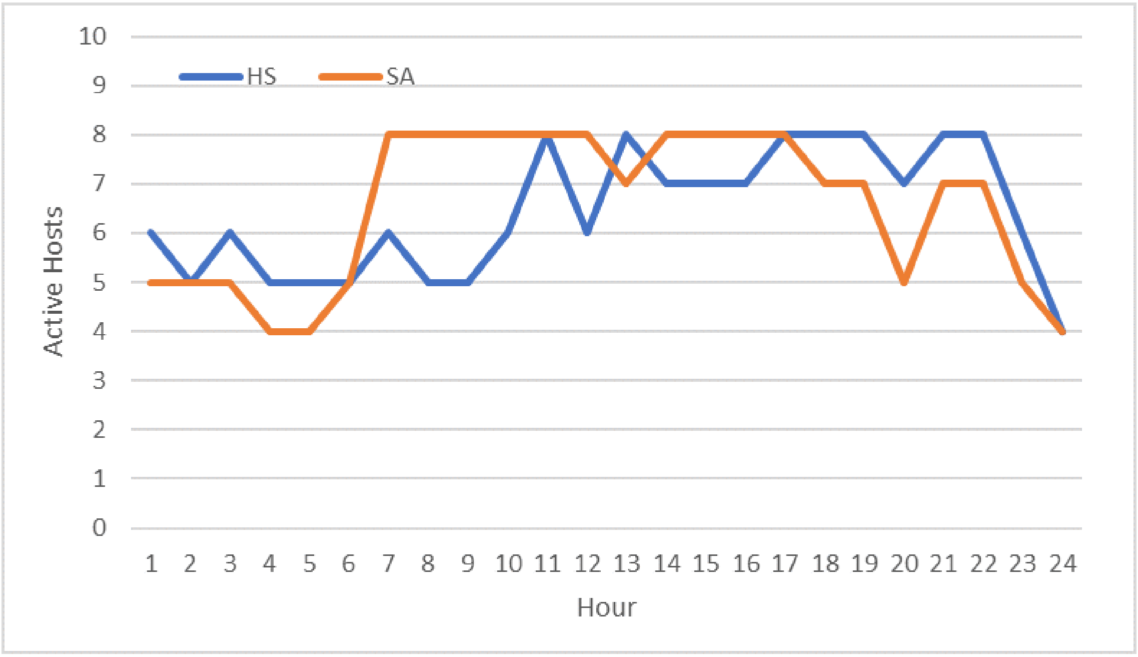


Figure 7.8: Comparison of number of active hosts

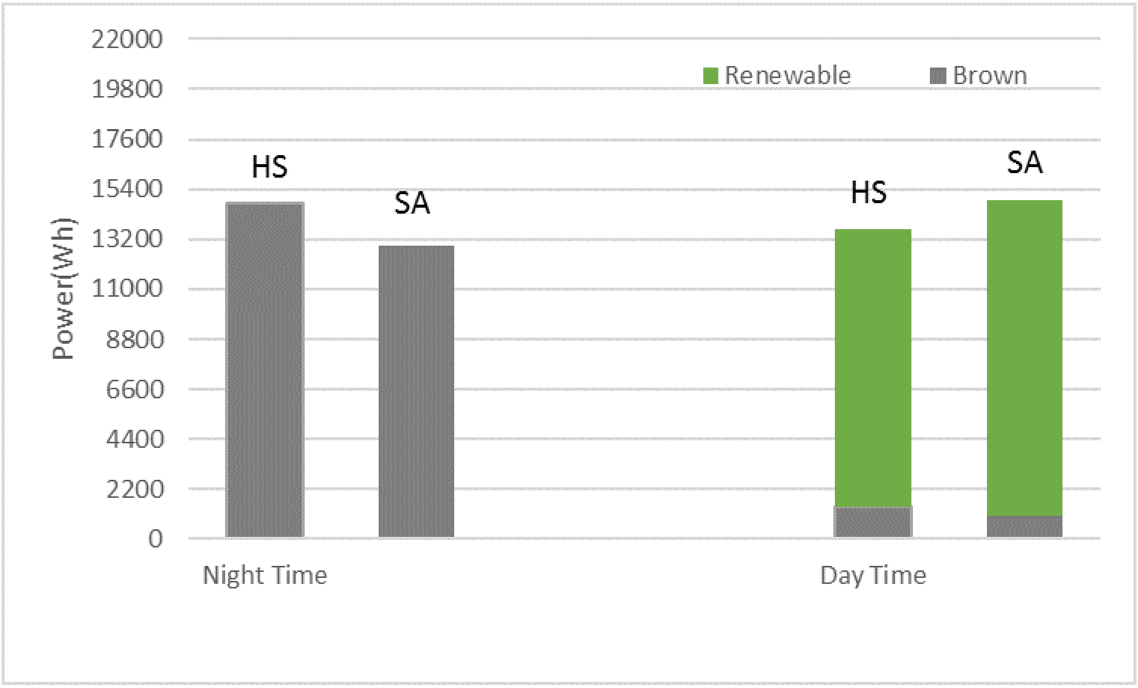


Figure 7.9: Comparison of different types of power usage

ing the night time (0:00 to 6:00 and 18:00 to 24:00), both approaches only use brown energy. Benefiting from proposed algorithms, our approach reduces the brown energy usage by 14%. During the daytime (7:00 to 17:00), both renewable energy and brown energy are used in two approaches. Our proposed approach consumes more total energy in the day time, while it uses 7% more renewable energy than the baseline from 12.9 KWh to 13.9 KWh, and brown energy usage is reduced by 43% from 1.4 KWh to 0.9 KWh.

In summary, experiments show that our proposed approach SA can improve the renewable energy usage for both interactive and batch workloads by applying brownout mechanism and deferring the execution of batch workloads. The brown energy usage is reduced 17%, and the renewable energy usage is improved 13%. Our proposed approach can switch more machines into low power mode when renewable energy is not sufficient while the QoS of workloads are also ensured.

7.8 Summary

The increased adoption of cloud computing and rapid growth in the deployment of data centers lead to the huge energy consumption issue. This needs a management platform which can manage the applications and energy in cloud data centers. Our self-adaptive approach for managing applications and harnessing renewable energy brings up many opportunities to optimize the energy efficiency problem in cloud computing environment. In this chapter, we proposed a multiple layer perspective model for interactive and batch workloads by considering renewable energy. We also introduced a self-adaptive and renewable energy-aware approach deriving from the perspective model. The proposed approach improves the usage of renewable energy and reduces the usage of brown energy while ensuring the QoS requirement of workloads. We proposed a solar radiation prediction method based on SVM to predict the solar power at Melbourne. The prediction method is integrated into our proposed approach. We apply brownout mechanism to dynamically deactivate/activate optional components in the system for interactive workloads and use a deferring algorithm to defer the execution of batch workloads to the time when renewable energy is available. VM consolidation and host scaling are also applied to reduce the number of active hosts.

We developed a prototype system to evaluate the performance of our proposed approach. In the prototype system, the physical resources are managed by OpenStack and the services are managed by Docker Swarm. We take advantage of the APIs from these platforms to monitor, manage, and provision the resources to services. The effectiveness of our proposed approach is showed through the experiments evaluation, including the workloads from real traces. The results show that our proposed approach is able to improve the usage of renewable energy while satisfying the constraints of workloads.

Chapter 8

Conclusions and Future Directions

This chapter provides a summary of research contributions on brownout-based approaches for energy efficient data centers. The summary of research outcomes and working experience extend the state-of-art energy efficient management for cloud data centers and also identify the challenges and future directions in the related area.

8.1 Summary of Contributions

Cloud computing is providing a paradigm for dynamically provisioning resources and delivering computing for applications as utility services as a pay-as-you-go basis. Providers like Amazon, Microsoft, IBM and Google have established data centers to support cloud applications around the world, and aimed to ensure that their services are flexible and suitable for the needs of the end-users.

However, energy consumed by the cloud data centers has currently become one of the major problems for the computing industry, since a typical data center can consume the energy as much as the power consumed by 25,000 households [48]. The growth and development of complex data-driven applications have promulgated the creation of huge data centers, which heightens the energy consumption. The energy consumed by servers constitutes more than half of total power consumption. The servers hosted in data centers dissipate more heat and need to be maintained in a fully air-conditioned and engineered environment. The carbon emissions caused by data centers also have impacts on the environment. Hence, reducing server energy consumption has become a significant topic in the IT industry.

It is an essential requirement for Cloud providers to reduce energy consumption, as it can both decrease operational costs and improve system reliability. Data centers can consume from 10 to 100 times more power per square foot than a typical office building. A large body of literature has focused on reducing energy consumption in cloud data centers, and the dominant categories for solving this problem are VM consolidation and Dynamic Voltage Frequency Scaling (DVFS). However, both of these approaches cannot function efficiently when the whole data center is overloaded.

One major reason of high energy consumption in cloud data centers is due to the inefficient resource utilization by application on servers. To utilize infrastructure resources in a more efficient manner, microservices are currently applied to build applications . Microservices are also referred to as a set of self-contained application components. The components encapsulate its logic and expose its functionality via interfaces to make them flexible to be deployed and replaced. With microservices or components, developers and user can benefit from their technological heterogeneity, resilience, scalability, ease of deployment, organizational alignment, composability and optimization for replicability, which can also achieve a more fine-grained control over the application resource usage.

A novel approach for managing resource usage in cloud computing systems is brownout, which was firstly introduced to cloud computing systems in 2014. In the field of brownout, the applications/services are extended to have two parts: mandatory and optional. The mandatory parts are desired to keep running all the time, such as the critical services in the systems including data-relevant services. The optional parts, on the other hand, are not necessary to be always active and can be deactivated temporarily to ensure the system performance in the case of flash crowds.

In this thesis, we have conducted a comprehensive literature review, proposed novel brownout-based approaches, and implemented prototype system to improve the state-of-art in these fields. Specifically, Chapter 1 introduced the background of the cloud computing system and why brownout is needed. Then the objectives of this thesis are summarized along with the research contributions.

Chapter 2 presented a review and taxonomy of brownout-based adaptive resources and applications management for cloud computing systems. The taxonomy is classified according to 5 phases: (1) application design, (2) workload scheduling, (3) monitoring, (4) brownout controller/dimmer design and (5) metrics. The taxonomy of each phase is summarized based on the different classification of brownout approaches. The review of existing brownout approaches is presented to identify the categories of each article and discuss the merits and limitations of articles. Furthermore, the comparison of the advantages and limitations of the surveyed brownout approaches are also made.

Chapter 3 to 7 focused on the specific brownout-based approaches to manage data center energy consumption. Chapter 3 proposed a scheduling algorithm for application components with brownout to balance the energy consumption and discount given to user. Chapter 4 presented a meta-heuristic algorithm based Markov Decision Process to select application components to improve the trade-offs between energy consumption and discount. Chapter 5 investigated an approach based on brownout and containers to save power consumption while ensuring QoS. Chapter 6 introduced a prototype system based on Docker Swarm to support brownout and evaluated its performance. Chapter 7 presented a self-adaptive approach for managing applications and renewable energy in sustainable cloud data centers.

Chapter 3 proposed the brownout enabled system model by taking application components into account. The application components can be either mandatory or optional based on the configurations of service provider. In the model, the brownout controller can dynamically disable/enable the optional components to reduce data center energy consumption. Meanwhile, the discount will be offered to users since functions are not fully provided. A brownout enabled algorithm as well as a number of policies to select components and investigate their effects on energy consumption and offered discount.

Chapter 4 introduced the brownout system model by deactivating optional components in applications or microservices temporarily. An algorithm based on brownout and approximate Markov Decision Process namely BMDP, to find the components should be deactivated was also proposed. The simulations based on real trace showed that BMDP can achieve better performance in energy and discount compared with baselines.

Chapter 5 presented a brownout-based architecture by deactivating optional containers in applications or microservices temporarily to reduce energy consumption. Under this architecture, an integrated approach to manage energy and brownout in container-based clouds was introduced. In addition, several policies to find suitable containers to deactivate were evaluated in a prototype system. The results showed that better performance can be achieved in energy consumption, response time and SLA violations than baselines.

Chapter 6 proposed the design and development of a software system based on brownout and containers for energy-efficient clouds, called BrownoutCon, which is a transparent system based on Docker Swarm for containers management and does not require to modify the default configurations of Docker Swarm via using its APIs. BrownoutCon can be customized for implementing brownout-based algorithms, which dynamically activates or deactivates containers to handle overloads and reduce energy consumption. The experiments conducted on Grid'5000 infrastructure showed that the brownout-based algorithms in BrownoutCon are able to reduce energy consumption while ensuring QoS.

Chapter 7 presented a multiple layer perspective model for both interactive and batch workloads by utilizing renewable energy to reduce carbon footprint. A self-adaptive approach derived from the perspective model was also proposed. The evaluations under our developed prototype system demonstrate that the proposed approach can improve the usage of renewable energy and reduce the usage of brown energy.

8.2 Future Directions

Although the significant progress of applying brownout to cloud computing systems has been achieved and adaptive management of resources and applications is improved, there are still some research gaps and challenges in this area to be further explored. This section provides some insights into some research gaps and promising future directions to broad the scenarios of brownout-based approaches and improve the resource management for cloud data centers.

8.2.1 Managing Different Resource Types

Managing multiple types of resources coordinately for cloud computing systems is important to maximize the resource usage. For resource management with brownout, current work mostly focus on the management of computation resources. More resource types, like memory, network and storage, need to be considered as parameters to form more comprehensive resource management. For example, brownout approach can be applied to manage data-intensive applications. By considering different resource types, more accurate energy model can also be modeled. The dimmer value based on multiple types of resources can also be calculated instead of computation resource only.

8.2.2 Exploring More Optimization Goals and Scenarios

Investigating into more diverse optimization goals can enhance the applicability of the brownout-based approach. Brownout has been applied to optimization goals including load balancing and energy efficiency. Besides these optimization goals, other goals such as more complicated cost-aware resource scheduling scenario, and more QoS metrics can be applied with brownout. Additionally, more scenarios, such as sustainable cloud computing systems with brownout can be investigated to reduce carbon emissions. Applying renewable energy is also an attractive scenario.

8.2.3 Developing Standard Benchmark

It is required to have a benchmark to test the performance of the new algorithm and compare it with other approaches having the same optimization objective. With the standard benchmark, the performance of proposed algorithms can be easily compared. Presently, there is a lack of the standard benchmark for performance evaluation of brownout-based approaches. Different workloads have been applied in articles, thus, it is still hard to compare the performance of different algorithms. The potential benchmarks can be derived from Facebook for batch workloads and Wikipedia for interactive workloads.

8.2.4 Investigating Distributed Controllers

Scalability should be considered when designing a resource scheduling approach, which supports to handle the increased number of requests in a smooth way. The centralized controller can be the bottleneck of the whole system, and scalability of the brownout controller can be improved by using distributed controllers. Therefore, distributed controllers design can be investigated. In distributed controllers, the dimmer values can be computed as varied values rather than a same dimmer value for all the controllers.

8.2.5 Integration with Machine Learning Techniques

The decision time and execution time of brownout-based algorithm to find the optional parts to be deactivated can be reduced by using machine learning methods. For example, based on response time constraints, requests can be clustered by machine learning algorithms (K-means) for further execution on the same type of machines to improve resource usage. Theory contributions are required in the related areas. The overhead and runtime complexity of heuristic and meta-heuristic approaches are not discussed in our surveyed works, which should be investigated.

8.2.6 New Cost Models

Service providers will be interested in integrated cost models. The existing cost models could be improved by considering more parameters, like electricity costs, budget, cooling costs and carbon emissions. Based on the cost models, the service provide can optimize their resource provisioning and maximize their profits. A more fine-grained cost model of brownout mechanism can also be investigated, such as the energy and time costs of the brownout activation/deactivation operations, and energy consumption of turning on/off servers.

8.2.7 Integration with Container Systems

Containers have provided a flexible ability to provide services with isolated function, and their quick start and stop operations enable brownout approach to work efficiently. Therefore, integrating brownout and containers is a promising direction to improve scheduling performance. The mature containers management platforms, like Apache Mesos, Kubernetes and Docker Swarm can be applied.

8.2.8 Applying Brownout with Other Application Models

As current works primarily focused on web applications, it is important to explore how brownout approaches can be applied in other application composition models such as Map-Reduce, bag of tasks application, and stream processing.

8.2.9 Integration with Existing Energy Efficient Approaches

It has been evaluated that brownout can be combined with VM consolidation to achieve better resource management effects. It is reasonable to expect better results when combining brownout with other existing energy efficiency techniques, such as DVFS. Some work based on the software-defined network (SDN) are optimizing energy consumption from network traffics perspective, which can also be combined with brownout, such as when the traffics are under the overloaded situation.

8.2.10 Shifting Workloads in Different Time Zones

Instead of considering a single data center, brownout approach for cloud computing systems can be extended to multiple data centers in different time zones. The knowledge pool in the perspective model can be used for tracking availability of the system when the system is scaled. In different time zones, the availability and amount of renewable energy can be varied. In the night, shifting the workloads to other time zones that have renewable energy can improve the usage of renewable energy and reduce the carbon footprint in the global view. The electricity costs in geographical data centers can also be considered to determine the workloads shift, thus the costs can also be optimized.

8.2.11 Integration with Fog/Edge Computing

Fog computing extends the cloud services to the edge of networks. The application design for Fog environment should leverage the real-time response from edge devices and use sufficient resources from the cloud simultaneously. In some fog computing applications, such as health monitoring and emergency response that require low latency, the delay that is caused by transferring data between the cloud and fog application can cause serious influence on their performance, such as no response at all. The brownout approach can be applied to reduce latency when there are network congestions. In addition, it is most likely that IoT/Edge devices have power constraints like the battery powered ones or may need to harness renewable energy like solar power, so the energy should be used efficiently. The brownout approach can support to optimize the energy usage for these devices by temporarily deactivating some optional application components.

8.3 Final Remarks

Cloud data centers are featured with the attractive characteristics of cloud computing, however, the energy consumption is a major concern for the IT industry. In this thesis, we investigated an approach called brownout to complement the existing approaches to reduce the energy consumption of cloud data centers. It proposed the architecture to enable brownout approach for cloud computing systems, algorithms for reducing energy while ensuring QoS, and prototype system to evaluate performance. The research outcome of this thesis is guidance to apply brownout to save power consumption of cloud data centers and investigate brownout in more scenarios for future directions.

Bibliography

[1] "Standard performance evaluation corporation." [Online]. Available: http://www.spec.org/power-ssj2008/results/res2010q2/

[2] Weaveshop-microservices-demo. [Online]. Available: https://github.com/microservices-demo/microservices-demo

[3] (2005) Rubbos: Bulletin board benchmark. [Online]. Available: http://jmob.ow2.org/rubbos.html

[4] (2009) Rubis. rubis: Rice university bidding system. [Online]. Available: http://rubis.ow2.org/

[5] (2014) 1998 world cup web site access logs - the internet traffic archive. [Online]. Available: http://ita.ee.lbl.gov/html/contrib/WorldCup.html

[6] (2014) Data center energy: Reducing your carbon footprint — data center knowledge. [Online]. Available: http://www.datacenterknowledge.com/archives/2014/12/17/undertaking-challenge-reduce-data-center-carbon-footprint

[7] (2016). [Online]. Available: http://www.afr.com/technology/banks-websites-down-as-wild-weather-knocks-out-amazon-web-services-20160605-gpc8ob

[8] (2017) Ansible is simple it automation. [Online]. Available: https://www.ansible.com/

[9] (2017) Apache jmeter - apache jmeter^{TM}. [Online]. Available: http://jmeter.apache.org/

[10] (2017) Docker compose file version 3 reference. [Online]. Available: https://docs.docker.com/compose/compose-file/

[11] (2017) Docker documentation — docker documentation. [Online]. Available: https://docs.docker.com/

[12] (2017) Weibo servers down after lu han announces new relationship. China Celebs. [Online]. Available: https://www.whatsonweibo.com/weibo-servers-lu-han-announces-new-relationship/

[13] (2018) Apache mesos. [Online]. Available: http://mesos.apache.org/

[14] (2018) Apache tomcat - welcome! [Online]. Available: http://tomcat.apache.org/

[15] (2018) Open source software for creating private and public clouds. [Online]. Available: https://www.openstack.org/

[16] (2018) Production-grade container orchestration - kubernetes. [Online]. Available: https://kubernetes.io/

[17] (2018) Swarm mode overview — docker documentation. [Online]. Available: https://docs.docker.com/engine/swarm/

[18] (2018) Vmware - official site. [Online]. Available: https://www.vmware.com/

[19] T. Adhikary, A. K. Das, M. A. Razzaque, M. Alrubaian, M. M. Hassan, and A. Alamri, "Quality of service aware cloud resource provisioning for social multimedia services and applications," *Multimedia Tools and Applications*, vol. 76, no. 12, pp. 14 485–14 509, 2017.

[20] A. M. Al-Qawasmeh, S. Pasricha, A. A. Maciejewski, and H. J. Siegel, "Power and thermal-aware workload allocation in heterogeneous data centers," *IEEE Transactions on Computers*, vol. 64, no. 2, pp. 477–491, 2015.

[21] F. Alvares, G. Delaval, E. Rutten, and L. Seinturier, "Language support for modular autonomic managers in reconfigurable software components," in *Proceedings of the 2017 IEEE International Conference on Autonomic Computing*. IEEE, 2017, pp. 271–278.

[22] A. S. Andrae and T. Edler, "On global electricity usage of communication technology: trends to 2030," *Challenges*, vol. 6, no. 1, pp. 117–157, 2015.

[23] P. Arcaini, E. Riccobene, and P. Scandurra, "Modeling and analyzing mape-k feedback loops for self-adaptation," in *Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2015, pp. 13–23.

[24] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya, "Dvfs-aware consolidation for energy-efficient clouds," in *Proceedings of the International Conference on Parallel Architecture and Compilation*. IEEE, 2015, pp. 494–495.

[25] M. S. Aslanpour, M. Ghobaei-Arani, and A. Nadjaran Toosi, "Auto-scaling web applications in clouds," *Journal of Network Computer Applications*, vol. 95, pp. 26–41, 2017.

[26] M. Avgerinou, P. Bertoldi, and L. Castellazzi, "Trends in data centre energy consumption under the european code of conduct for data centre energy efficiency," *Energies*, vol. 10, no. 10, p. 1470, 2017.

[27] K. Y. Bae, H. S. Jang, and D. K. Sung, "Hourly solar irradiance prediction based on support vector machine and its error analysis," *IEEE Transactions on Power Systems*, vol. 32, no. 2, pp. 935–945, 2017.

[28] L. Baresi, S. Guinea, G. Quattrocchi, and D. A. Tamburri, "Microcloud: A container-based solution for efficient resource management in the cloud," in *Proceeding of the IEEE International Conference on Smart Cloud*. IEEE, 2016, pp. 218–223.

[29] T. Bawden. (2016) Global warming: Data centres to consume three times as much energy in next decade, experts warn. [Online]. Available: http://www.independent.co.uk/environment/global-warming-data-centres-to-consume-three-times-as-much-energy-in-next-decade-experts-warn-a6830086.html

[30] S. Belaid and A. Mellit, "Prediction of daily and mean monthly global solar radiation using support vector machine in an arid climate," *Energy Conversion and Management*, vol. 118, pp. 105–118, 2016.

[31] R. Bellman, "Dynamic programming and lagrange multipliers," *Proceedings of the National Academy of Sciences*, vol. 42, no. 10, pp. 767–769, 1956.

[32] A. Belog1azov and R. Buyya, "Managing overloaded hosts for dynamic consolidation of virtual machines in cloud data centers under quality of service constraints," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 7, pp. 1366–1379, 2013.

[33] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[34] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[35] ——, "Openstack neat: a framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 5, pp. 1310–1333, 2015.

[36] J. Bi, H. Yuan, W. Tan, M. Zhou, Y. Fan, J. Zhang, and J. Li, "Application-aware dynamic fine-grained resource provisioning in a virtualized cloud data center," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 2, pp. 1172–1184, April 2017.

[37] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. cambridge university press, 2005.

[38] D. Boru, D. Kliazovich, F. Granelli, P. Bouvry, and A. Y. Zomaya, "Energy-efficient data replication in cloud computing datacenters," *Cluster computing*, vol. 18, no. 1, pp. 385–402, 2015.

[39] R. Buyya, R. N. Calheiros, J. Son, A. V. Dastjerdi, and Y. Yoon, "Software-defined cloud computing: Architectural elements and open challenges," in *Proceedings of the 2014 International Conference on Advances in Computing, Communications and Informatics*, 2014, pp. 1–12.

[40] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *Proceedings of the International Conference on High Performance Computing and Simulation*, 2009, pp. 1–11.

[41] R. Buyya, C. S. Yeo, and S. Venugopal, "Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities," in *Proceedings of the 10th IEEE International Conference on High Performance Computing and Communications*, 2008, pp. 5–13.

[42] 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.

[43] Q. Chen, J. Chen, B. Zheng, J. Cui, and Y. Qian, "Utilization-based vm consolidation scheme for power efficiency in cloud data centers," in *Proceedings of the 2015 IEEE International Conference on Communication Workshop (ICCW)*. IEEE, 2015, pp. 1928–1933.

[44] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Elastic power-aware resource provisioning of heterogeneous workloads in self-sustainable datacenters," *IEEE Transactions on Computer*, vol. 65, no. 2, pp. 508–521, 2016.

[45] D. Cheng, X. Zhou, P. Lama, M. Ji, and C. Jiang, "Energy efficiency aware task assignment with dvfs in heterogeneous hadoop clusters," *IEEE Transactions on Parallel & Distributed Systems*, no. 1, pp. 1–1, 2018.

[46] B. Clement. (2017) Rightscale 2017 state of the cloud report uncovers cloud adoption trends. [Online]. Available: https://www.rightscale.com/press-releases/

[47] A. Corradi, M. Fanelli, and L. Foschini, "Vm consolidation: A real case based on openstack cloud," *Future Generation Computer Systems*, vol. 32, pp. 118–127, 2014.

[48] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 732–794, 2016.

[49] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, 2013, pp. 1–32.

[50] P. Delforge. (2014) Data center efficiency assessment - scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. [Online]. Available: https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf

[51] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini, "Coscale: Coordinating cpu and memory system dvfs in server systems," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2012, pp. 143–154.

[52] S. Deng, H. Wu, W. Tan, Z. Xiang, and Z. Wu, "Mobile service selection for composition: An energy consumption perspective," *IEEE Transactions on Automation Science and Engineering*, vol. 14, no. 3, pp. 1478–1490, July 2017.

[53] D. Desmeurs, *Algorithms for Event-Driven Application Brownout*. Master Thesis, Umea University, 2015.

[54] D. Desmeurs, C. Klein, A. V. Papadopoulos, and J. Tordsson, "Event-driven application brownout: Reconciling high utilization and low tail response times," in *Proceedings of the 2015 International Conference on Cloud and Autonomic Computing*, 2015, pp. 1–12.

[55] Docker. (2017) Docker documentation — docker documentation. [Online]. Available: https://docs.docker.com/

[56] W. Dou, X. Xu, S. Meng, X. Zhang, C. Hu, S. Yu, and J. Yang, "An energy-aware virtual machine scheduling method for service qos enhancement in clouds over big data," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 14, pp. 1–20, 2017.

[57] S. Dupont, J. Lejeune, F. Alvares, and T. Ledoux, "Experimental analysis on autonomic strategies for cloud elasticity," in *Proceedings of the 2015 IEEE International Conference on Cloud and Autonomic Computing*, 2015, pp. 81–92.

[58] J. Dürango, M. Dellkrantz, M. Maggio, C. Klein, A. V. Papadopoulos, F. Hernández-Rodriguez, E. Elmroth, and K.-E. Årzén, "Control-theoretical load-balancing for cloud applications with brownout," in *Proceedings of the 2014 IEEE 53rd Annual Conference on Decision and Control*, 2014, pp. 5320–5327.

[59] A. EC2. (2018) Amazon web services. [Online]. Available: https://aws.amazon.com/ec2/

[60] T. Everts. (2016) Google: 53than 3 seconds to load. [Online]. Available: https://www.soasta.com/blog/google-mobile-web-performance-study/

[61] Q. Fang, J. Wang, and Q. Gong, "Qos-driven power management of data centers via model predictive control," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 4, pp. 1557–1566, Oct 2016.

[62] M. H. Ferdaus, M. Murshed, R. N. Calheiros, and R. Buyya, "Virtual machine consolidation in cloud data centers using aco metaheuristic," in *Proceedings of the European Conference on Parallel Processing*. Springer, 2014, pp. 306–317.

[63] K. Gai, M. Qiu, and H. 2hao, "Energy-aware task assignment for mobile cyber-enabled applications in heterogeneous cloud computing," *Journal of Parallel and Distributed Computing*, vol. 111, pp. 126–135, 2018.

[64] K. Gai, M. Qiu, and H. Zhao, "Cost-aware multimedia data allocation for heterogeneous memory using genetic algorithm in cloud computing," *IEEE Transactions on Cloud Computing*, 2016.

[65] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *Journal of Network and Computer Applications*, vol. 59, pp. 46–54, 2016.

[66] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," in *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1. ACM, 2013, pp. 51–64.

[67] Grid5000. (2017) Grid5000:home. [Online]. Available: https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home

[68] P. Habibi, M. Mokhtari, and M. Sabaei, "Qrve: Qos-aware routing and energy-efficient vm placement for software-defined datacenter networks," in *Telecommunications (IST), 2016 8th International Symposium on*. IEEE, 2016, pp. 533–539.

[69] D. Han and T. Shu, "Thermal-aware energy-efficient task scheduling for dvfs-enabled data centers," in *Proceedings of the 2015 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2015, pp. 536–540.

[70] G. Han, W. Que, G. Jia, and L. Shu, "An efficient virtual machine consolidation scheme for multimedia cloud computing," *Sensors*, vol. 16, no. 2, p. 246, 2016.

[71] Z. Han, H. Tan, G. Chen, R. Wang, Y. Chen, and F. C. M. Lau, "Dynamic virtual machine management via approximate markov decision process," in *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, April 2016, pp. 1–9.

[72] V. Hanumaiah and S. Vrudhula, "Energy-efficient operation of multicore processors by dvfs, task migration, and active cooling," *IEEE Transactions on Computers*, vol. 63, no. 2, pp. 349–360, 2014.

[73] M. S. Hasan, F. Alvares, and T. Ledoux, "Gpaascaler: Green energy aware platform scaler for interactive cloud application," in *Proceedings of the 10th ACM International Conference on Utility and Cloud Computing*, 2017, pp. 79–89.

[74] M. S. Hasan, F. Alvares, T. Ledoux, and J.-L. Pazat, "Investigating energy consumption and performance trade-off for interactive cloud application," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 113–126, 2017.

[75] M. S. Hasan, F. A. de Oliveira, T. Ledoux, and J.-L. Pazat, "Enabling green energy awareness in interactive cloud application," in *Proceedings of the 2016 IEEE International Conference on Cloud Computing Technology and Science*, 2016, pp. 414–422.

[76] S. Homsi, S. Liu, G. A. Chaparro-Baquero, O. Bai, S. Ren, and G. Quan, "Workload consolidation for cloud data centers with guaranteed qos using request reneging," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2103–2116, 2017.

[77] 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.

[78] D. G. D. L. Iglesia and D. Weyns, "Mape-k formal templates to rigorously design behaviors for self-adaptive systems," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 10, no. 3, pp. 1–31, 2015.

[79] N. Irwin. (2013) These 12 technologies will drive our economic future. [Online]. Available: https://www.washingtonpost.com/news/wonk/wp/2013/05/24/these-12-technologies-will-drive-our-economic-future/?utm_term=.e5ad3815c7eb

[80] N. J. Kansal and I. Chana, "Cloud load balancing techniques: A step towards green computing," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 1, pp. 238–246, 2012.

[81] T. Kaur and I. Chana, "Energy efficiency techniques in cloud computing: A survey and taxonomy," *ACM Computing Surveys (CSUR)*, vol. 48, no. 2, pp. 1–46, 2015.

[82] M. E. Khanouche, Y. Amirat, A. Chibani, M. Kerkar, and A. Yachir, "Energy-centered and qos-aware services selection for internet of things," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 3, pp. 1256–1269, 2016.

[83] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of virtual machines for real-time cloud services," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 13, pp. 1491–1505, 2011.

[84] S. Kim, S. Park, Y. Kim, S. Kim, and K. Lee, "Vnf-eq: dynamic placement of virtual network functions for energy efficiency and qos guarantee in nfv," *Cluster Computing*, vol. 20, no. 3, pp. 2107–2117, 2017.

[85] 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*, 2014, pp. 700–711.

[86] C. Klein, A. V. Papadopoulos, M. Dellkrantz, J. Dürango, M. Maggio, K.-E. Årzén, F. Hernández-Rodriguez, and E. Elmroth, "Improving cloud service resilience using brownout-aware load-balancing," in *Proceedings of the 2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, 2014, pp. 31–40.

[87] F. Kong and X. Liu, "A survey on green-energy-aware power management for datacenters," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, p. 30, 2015.

[88] Z. Kozhirbayev and R. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Generation Computer Systems*, vol. 68, pp. 175–182, 2017.

[89] H. Li, G. Zhu, Y. Zhao, Y. Dai, and W. Tian, "Energy-efficient and qos-aware model based resource consolidation in cloud data centers," *Cluster Computing*, pp. 1–11, 2017.

[90] X. Li, X. Jiang, P. Garraghan, and Z. Wu, "Holistic energy and failure aware workload scheduling in cloud datacenters," *Future Generation Computer Systems*, vol. 78, pp. 887–900, 2018.

[91] Z. Li, C. Yan, X. Yu, and N. Yu, "Bayesian network-based virtual machines consolidation method," *Future Generation Computer Systems*, vol. 69, pp. 75–87, 2017.

[92] B. Liao, J. Yu, T. Zhang, G. Binglei, S. Hua, and C. Ying, "Energy-efficient algorithms for distributed storage system based on block storage structure reconfiguration," *Journal of Network and Computer Applications*, vol. 48, pp. 71–86, 2015.

[93] J. M. Lima. (2017) Data centres of the world will consume 1/5 of earth's power by 2025. [Online]. Available: https://data-economy.com/data-centres-world-will-consume-1-5-earths-power-2025/

[94] H. Liu, B. Liu, L. T. Yang, M. Lin, Y. Deng, K. Bilal, and S. U. Khan, "Thermal-aware and dvfs-enabled big data task scheduling for data centers," *IEEE Transactions on Big Data*, vol. 4, no. 2, pp. 177–190, 2018.

[95] K. Liu, K. Aida, S. Yokoyama, and Y. Masatani, "Flexible container-based computing platform on cloud for scientific workflows," in *Proceedings of International Conference on Cloud Computing Research and Innovations*. IEEE, 2016, pp. 56–63.

[96] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser, "Renewable and cooling aware workload management for sustainable data centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 175–186.

[97] ——, "Renewable and cooling aware workload management for sustainable data centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1. ACM, 2012, pp. 175–186.

[98] Z. Liu, M. Lin, A. Wierman, S. Low, and L. L. Andrew, "Greening geographical load balancing," *IEEE/ACM Transactions on Networking*, vol. 23, no. 2, pp. 657–671, 2015.

[99] T. Lorido-Botran, J. Miguel-Alonso, and J. A. Lozano, "A review of auto-scaling techniques for elastic applications in cloud environments," *Journal of Grid Computing*, vol. 12, no. 4, pp. 559–592, 2014.

[100] A. Luzzardi. Scale testing docker swarm to 30,000 containers - docker blog. [Online]. Available: https://blog.docker.com/2015/11/scale-testing-docker-swarm-30000-containers/

[101] M. Maggio, C. Klein, and K.-E. Årzén, "Control strategies for predictable brownouts in cloud computing," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 689–694, 2014.

[102] Y. Mansouri, A. N. Toosi, and R. Buyya, "Data storage management in cloud environments: Taxonomy, survey, and future directions," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–51, 2017.

[103] T. Maqsood, N. Tziritas, T. Loukopoulos, S. A. Madani, S. U. Khan, and C.-Z. Xu, "Leveraging on deep memory hierarchies to minimize energy consumption and data access latency on single-chip cloud computers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 154–166, 2017.

[104] T. Mastelic, A. Oleksiak, H. Claussen, I. Brandic, J.-M. Pierson, and A. V. Vasilakos, "Cloud computing: Survey on energy efficiency," *ACM Computing Surveys (CSUR)*, vol. 47, no. 2, pp. 1–36, 2015.

[105] C. Mastroianni, M. Meo, and G. Papuzzo, "Probabilistic consolidation of virtual machines in self-organizing cloud data centers," *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 215–228, 2013.

[106] D. Meisner, B. T. Gold, and T. F. Wenisch, "Powernap: eliminating server idle power," in *ACM sigplan notices*, vol. 44, no. 3. ACM, 2009, pp. 205–216.

[107] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[108] A. S. Milani and N. J. Navimipour, "Load balancing mechanisms and techniques in the cloud environments: Systematic literature review and future trends," *Journal of Network and Computer Applications*, vol. 71, pp. 86–98, 2016.

[109] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *Journal of Network and Computer Applications*, vol. 64, pp. 229–238, 2016.

[110] F. F. Moghaddam and M. Cheriet, "Sustainability-aware cloud computing using virtual carbon tax," *arXiv preprint arXiv:1510.05182*, 2015.

[111] M. A. H. Monil and R. M. Rahman, "Fuzzy logic based energy aware vm consolidation," in *Proceedings of the International Conference on Internet and Distributed Computing Systems*. Springer, 2015, pp. 31–38.

[112] J. D. Moore, J. S. Chase, P. Ranganathan, and R. K. Sharma, "Making scheduling" cool": Temperature-aware workload placement in data centers." in *Proceedings of the USENIX annual technical conference, General Track*, 2005, pp. 61–75.

[113] G. A. Moreno, *Adaptation Timing in Self-Adaptive Systems*. PhD Thesis, Carnegie Mellon University, 2017.

[114] G. A. Moreno, J. Cámara, D. Garlan, and B. Schmerl, "Proactive self-adaptation under uncertainty: a probabilistic model checking approach," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 1–12.

[115] ——, "Efficient decision-making under uncertainty for proactive self-adaptation," in *Proceedings of the 2016 IEEE International Conference on Autonomic Computing*, 2016, pp. 147–156.

[116] S. Newman, *Building Microservices*. " O'Reilly Media, Inc.", 2015.

[117] T. H. Nguyen, M. D. Francesco, and A. Yla-Jaaski, "Virtual machine consolidation with multiple usage prediction for energy-efficient cloud data centers," *IEEE Transactions on Services Computing*, pp. 1–14, 2018.

[118] V. Nikolov, S. Kächele, F. J. Hauck, and D. Rautenbach, "Cloudfarm: An elastic cloud platform with flexible and adaptive resource management," in *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 547–553.

[119] C. Pahl, A. Brogi, J. Soldani, and P. Jamshidi, "Cloud container technologies: a state-of-the-art review," *IEEE Transactions on Cloud Computing*, 2017.

[120] A. Pandey, G. A. Moreno, J. Cámara, and D. Garlan, "Hybrid planning for decision making in self-adaptive systems," in *Proceedings of the 2016 IEEE 10th International Conference on Self-Adaptive and Self-Organizing Systems*, 2016, pp. 130–139.

[121] R. Panwar and B. Mallick, "Load balancing in cloud computing using dynamic load management algorithm," in *Proceedings of the 2015 IEEE International Conference on Green Computing and Internet of Things*, 2015, pp. 773–778.

[122] K. Park and V. S. Pai, "Comon: a mostly-scalable monitoring system for planetlab," *ACM SIGOPS Operating Systems Review*, vol. 40, no. 1, pp. 65–74, 2006.

[123] J. E. Pecero, H. J. F. Huacuja, P. Bouvry, A. A. S. Pineda, M. C. L. Locés, and J. J. G. Barbosa, "On the energy optimization for precedence constrained applications using local search algorithms," in *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS)*, 2012, pp. 133–139.

[124] M. Pedram, "Energy-efficient datacenters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 10, pp. 1465–1484, 2012.

[125] I. Pietri and R. Sakellariou, "Energy-aware workflow scheduling using frequency scaling," in *Proceedings of the 2014 43rd International Conference on Parallel Processing Workshops*. IEEE, 2014, pp. 104–113.

[126] D. Pompili, A. Hajisami, and T. X. Tran, "Elastic resource utilization framework for high capacity and energy efficiency in cloud ran," *IEEE Communications Magazine*, vol. 54, no. 1, pp. 26–32, 2016.

[127] A. Rahman, X. Liu, and F. Kong, "A survey on geographic load balancing based data center power management in the smart grid environment," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 214–233, 2014.

[128] M. Randles, D. Lamb, and A. Taleb-Bendiab, "A comparative study into distributed load balancing algorithms for cloud computing," in *Proceedings of the 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops*, 2010, pp. 551–556.

[129] M. A. Salehi, P. R. Krishna, K. S. Deepak, and R. Buyya, "Preemption-aware energy management in virtualized data centers," in *Proceedings of the 5th International Conference on Cloud Computing (CLOUD)*, 2012, pp. 844–851.

[130] A. M. Sampaio, J. G. Barbosa, and R. Prodan, "Piasa: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers," *Simulation Modelling Practice and Theory*, vol. 57, pp. 142–160, 2015.

[131] M. Shahrad, C. Klein, L. Zheng, M. Chiang, E. Elmroth, and D. Wentzlaff, "Incentivizing self-capping to increase cloud utilization," in *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 2017, pp. 52–65.

[132] Y. Sharma, B. Javadi, and W. Si, "On the reliability and energy efficiency in cloud computing," *Parallel and Distributed Computing*, vol. 27, p. 111, 2015.

[133] J. Shuja, A. Gani, S. Shamshirband, R. W. Ahmad, and K. Bilal, "Sustainable cloud data centers: a survey of enabling techniques and technologies," *Renewable and Sustainable Energy Reviews*, vol. 62, pp. 195–214, 2016.

[134] S. Singh and I. Chana, "Earth: Energy-aware autonomic resource scheduling in cloud computing," *Journal of Intelligent and Fuzzy Systems*, vol. 30, no. 3, pp. 1581–1600, 2016.

[135] ——, "Qos-aware autonomic resource management in cloud computing: a systematic review," *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–46, 2016.

[136] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Sla-aware and energy-efficient dynamic overbooking in sdn-based cloud data centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 76–89, 2017.

[137] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley and Sons, 2009, vol. 74.

[138] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *Proceedings of the 2015 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[139] Q. Tang, S. K. S. Gupta, and G. Varsamopoulos, "Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 11, pp. 1458–1472, 2008.

[140] F. Teng, L. Yu, T. Li, D. Deng, and F. Magoulès, "Energy efficiency of vm consolidation in iaas clouds," *The Journal of Supercomputing*, pp. 1–28, 2016.

[141] P. R. Theja and S. Babu, "Evolutionary computing based on qos oriented energy efficient vm consolidation scheme for large scale cloud data centers," *Cybernetics and Information Technologies*, vol. 16, no. 2, pp. 97–112, 2016.

[142] L. Tomás, C. Klein, J. Tordsson, and F. Hernández-Rodríguez, "The straw that broke the camel's back: safe cloud overbooking with application brownout," in *Proceedings of the 2014 IEEE International Conference on Cloud and Autonomic Computing*, 2014, pp. 151–160.

[143] A. N. Toosi, C. Qu, M. D. de Assunção, and R. Buyya, "Renewable-aware geographical load balancing of web applications for sustainable data centers," *Journal of Network and Computer Applications*, vol. 83, pp. 155–168, 2017.

[144] A. N. Toosi, K. Vanmechelen, K. Ramamohanarao, and R. Buyya, "Revenue maximization with optimal capacity control in infrastructure as a service cloud markets," *IEEE Transactions on Cloud Computing*, vol. 3, no. 3, pp. 261–274, 2015.

[145] G. L. Valentini, W. Lassonde, S. U. Khan, N. Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej *et al.*, "An overview of energy efficiency techniques in cluster computing systems," *Cluster Computing*, vol. 16, no. 1, pp. 3–15, 2013.

[146] G. Von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-aware scheduling of virtual machines in dvfs-enabled clusters," in *Proceedings of the IEEE International Conference on Cluster Computing and Workshops*, 2009, pp. 1–10.

[147] J. Wang, C. Huang, Q. Liu, K. He, J. Wang, P. Li, and X. Jia, "An optimization vm deployment for maximizing energy utility in cloud environment," in *Proceedings of the International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 2014, pp. 400–414.

[148] S. Wang, A. Zhou, C.-H. Hsu, X. Xiao, and F. Yang, "Provision of data-intensive services through energy-and qos-aware virtual machine placement in national cloud data centers," *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 2, pp. 290–300, 2016.

[149] D. Weerasiri, M. C. Barukh, B. Benatallah, Q. Z. Sheng, and R. Ranjan, "A taxonomy and survey of cloud resource orchestration techniques," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–41, 2017.

[150] B. Whitehead, D. Andrews, A. Shah, and G. Maidment, "Assessing the environmental impact of data centres part 1: Background, energy use and metrics," *Building and Environment*, vol. 82, pp. 151 – 159, 2014.

[151] Wikibench. (2017). [Online]. Available: http://www.wikibench.eu/wiki/2007-10/

[152] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li, "Cloud computing resource scheduling and a survey of its evolutionary approaches," *ACM Computing Surveys (CSUR)*, vol. 47, no. 4, pp. 1–33, 2015.

[153] Y. Zhang, M. A. Laurenzano, J. Mars, and L. Tang, "Smite: Precise qos prediction on real-system smt processors to improve utilization in warehouse scale computers," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 406–418.

[154] T. Zhao, W. Zhang, H. Zhao, and Z. Jin, "A reinforcement learning-based framework for the generation and evolution of adaptation rules," in *Proceedings of the 2017 IEEE International Conference on Autonomic Computing*, 2017, pp. 103–112.

[155] 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 Conference on Computer Communications (INFOCOM)*, 2014, pp. 2598–2606.

[156] Y. Zuo, F. Tao, and A. Y. Nee, "An internet of things and cloud-based approach for energy consumption evaluation and analysis for a product," *International Journal of Computer Integrated Manufacturing*, vol. 31, no. 4-5, pp. 337–348, 2018.