

6-15-2016


# Sustainable Resource Management for Cloud Data Centers

A. S. M. Hasan Mahmud

*Florida International University, amahm008@fiu.edu*

**DOI:** 10.25148/etd.FIDC000693

Follow this and additional works at: <https://digitalcommons.fiu.edu/etd>

 Part of the [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), [Systems Architecture Commons](#), and the [Theory and Algorithms Commons](#)

---

## Recommended Citation

Mahmud, A. S. M. Hasan, "Sustainable Resource Management for Cloud Data Centers" (2016). *FIU Electronic Theses and Dissertations*. 2634.

<https://digitalcommons.fiu.edu/etd/2634>

This work is brought to you for free and open access by the University Graduate School at FIU Digital Commons. It has been accepted for inclusion in FIU Electronic Theses and Dissertations by an authorized administrator of FIU Digital Commons. For more information, please contact [dcc@fiu.edu](mailto:dcc@fiu.edu).

FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SUSTAINABLE RESOURCE MANAGEMENT FOR CLOUD DATA CENTERS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

A. S. M. Hasan Mahmud

2016

To: Interim Dean Ranu Jung  
College of Engineering and Computing

This dissertation, written by A. S. M. Hasan Mahmud, and entitled Sustainable Resource Management for Cloud Data Centers, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

---

Shaolei Ren

---

Deng Pan

---

Jason Liu

---

Leonardo Bobadilla

---

Gang Quan

---

S. S. Iyengar, Major Professor

Date of Defense: June 15, 2016

The dissertation of A. S. M. Hasan Mahmud is approved.

---

Interim Dean Ranu Jung  
College of Engineering and Computing

---

Andrés G. Gil  
Vice President for Research and Economic Development  
and Dean of the University Graduate School

Florida International University, 2016

© Copyright 2016 by A. S. M. Hasan Mahmud

All rights reserved.

DEDICATION

To my parents.

## ACKNOWLEDGMENTS

I could not be more grateful to my committee member Dr. Shaolei Ren who was my supervisor for the first three years. He kept in touch even after leaving the university and consistently helped me in completing the ongoing research. Throughout the years, his enthusiasm, support, and vision gave me the necessary strength to proceed through the doctoral program and finish this dissertation.

I would like to express my sincere gratitude to my supervisor Dr. S. S. Iyengar for his help and support during the last years of my doctoral program. I am thankful to my committee members Dr. Jason Liu, Dr. Deng Pan, Dr. Leonardo Bobadilla, and Dr. Gang Quan for their valuable suggestions to improve this dissertation. Dr. Yuxiong He from Microsoft Research has provided valuable insights while completing the final part of this dissertation. I am grateful for her active support which improved this dissertation considerably.

I like to thank my family members who supported me over the last five years. My mother Shamshun Nahar always encouraged me to pursue higher study since I was a little boy. This dissertation would not be even possible without her guidance, love and support. I could not thank my sisters Shanjida and Thanjida enough for their unconditional support over the years. I am also thankful to my brother-in-law Mohammad Atiqul Islam, who also happens to be my colleague. We worked together on many projects and my discussion with him regarding the research problems greatly benefited me in countless scenarios.

ABSTRACT OF THE DISSERTATION  
SUSTAINABLE RESOURCE MANAGEMENT FOR CLOUD DATA CENTERS

by

A. S. M. Hasan Mahmud

Florida International University, 2016

Miami, Florida

Professor S. S. Iyengar, Major Professor

In recent years, the demand for data center computing has increased significantly due to the growing popularity of cloud applications and Internet-based services. Today's large data centers host hundreds of thousands of servers and the peak power rating of a single data center may even exceed 100MW. The combined electricity consumption of global data centers accounts for about 3% of worldwide production, raising serious concerns about their carbon footprint. The utility providers and governments are consistently pressuring data center operators to reduce their carbon footprint and energy consumption. While these operators (e.g., Apple, Facebook, and Google) have taken steps to reduce their carbon footprints (e.g., by installing on-site/off-site renewable energy facility), they are aggressively looking for new approaches that do not require expensive hardware installation or modification.

This dissertation focuses on developing algorithms and systems to improve the sustainability in data centers without incurring significant additional operational or setup costs. In the first part, we propose a provably-efficient resource management solution for a self-managed data center to cap and reduce the carbon emission while maintaining satisfactory service performance. Our solution reduces the carbon emission of a self-managed data center to net-zero level and achieves carbon neutrality. In the second part, we consider minimizing the carbon emission in a hybrid data center infrastructure that includes geographically distributed self-managed and colocation data centers. This segment

identifies and addresses the challenges of resource management in a hybrid data center infrastructure and proposes an efficient distributed solution to optimize the workload and resource allocation jointly in both self-managed and colocation data centers. In the final part, we explore sustainable resource management from cloud service users' point of view. A cloud service user purchases computing resources (e.g., virtual machines) from the service provider and does not have direct control over the carbon emission of the service provider's data center. Our proposed solution encourages a user to take part in sustainable (both economical and environmental) computing by limiting its spending on cloud resource purchase while satisfying its application performance requirements.



## TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION . . . . .	1
1.1 Motivation . . . . .	1
1.2 Problem Definition and Contributions . . . . .	3
1.2.1 Sustainability in a self-managed data center . . . . .	3
1.2.2 Sustainability in a hybrid data center infrastructure . . . . .	5
1.2.3 Sustainability from a cloud service user’s point of view . . . . .	6
1.3 Related Publications . . . . .	8
1.4 Outline of the Dissertation . . . . .	9
2. RELATED WORK . . . . .	10
2.1 Dynamic Server Provisioning . . . . .	10
2.2 Geographical Load Balancing (GLB) . . . . .	12
2.3 Colocation Data Center . . . . .	15
2.4 Autoscaling of Virtualized Resources . . . . .	17
2.4.1 Proactive autoscaling . . . . .	18
2.4.2 Reactive autoscaling . . . . .	19
2.5 How Our Work is Different? . . . . .	20
2.5.1 Sustainability in a self-managed data center . . . . .	20
2.5.2 Sustainability in a hybrid data center infrastructure . . . . .	21
2.5.3 Sustainability from a cloud service user’s point of view . . . . .	22
3. SUSTAINABILITY IN A SELF-MANAGED DATA CENTER . . . . .	25
3.1 Background . . . . .	25
3.2 Model . . . . .	28
3.2.1 Workloads . . . . .	28
3.2.2 Data center . . . . .	30
3.2.3 Demand-responsive electricity price . . . . .	31
3.2.4 Renewable energy . . . . .	33
3.3 Algorithm Design and Analysis . . . . .	35
3.3.1 Problem formulation . . . . .	35
3.3.2 CNDC . . . . .	37
3.3.3 Performance analysis . . . . .	39
3.4 Simulation Study . . . . .	46
3.4.1 Data sets . . . . .	46
3.4.2 Impact of $V$ . . . . .	49
3.4.3 Comparison with prediction-based method . . . . .	51
3.4.4 Comparison with price unaware methods . . . . .	53
3.4.5 Sensitivity Study . . . . .	54
3.5 System Experiment . . . . .	56
3.5.1 Setup . . . . .	57

3.5.2	Results	58
3.6	Summary	60
4.	SUSTAINABILITY IN A HYBRID DATA CENTER INFRASTRUCTURE	62
4.1	Background	62
4.2	Model and Problem Formulation	65
4.2.1	Data center and propagation delay	66
4.2.2	Power consumption and carbon footprints	68
4.2.3	Electricity Cost	70
4.2.4	Problem formulation	71
4.3	Algorithm Design and Analysis	72
4.3.1	Solution to P21	72
4.3.2	Applying ADMM to solve P21	74
4.3.3	Performance of CAGE	77
4.4	Simulation Study	78
4.4.1	Data and settings	80
4.4.2	Benchmarks comparison	81
4.4.3	Impact of carbon-cost parameter $\sigma$	84
4.4.4	Renewable energy	85
4.5	System Experiment	86
4.5.1	Test-bed setup	86
4.5.2	Settings and results	88
4.6	Summary	89
5.	SUSTAINABILITY FROM A CLOUD SERVICE USER'S POINT OF VIEW	90
5.1	Background	90
5.2	Model and Problem Formulation	94
5.2.1	Model	94
5.2.2	Problem Formulation	95
5.3	Algorithm Design and Analysis	96
5.3.1	Obtaining Inputs to BATS	97
5.3.2	BATS	98
5.3.3	Performance of BATS	100
5.3.4	Discussion	104
5.3.5	Reserved Instances	105
5.4	System Implementation	108
5.4.1	Architecture Overview	110
5.4.2	Scheduler	111
5.5	System Experiment	112
5.5.1	Experimental Setup	112
5.5.2	Experimental Results	115
5.6	Simulation Study	121
5.6.1	Simulator Overview and Validation	122

5.6.2	Experimental Setup . . . . .	123
5.6.3	Optimizing Average and Tail Delay . . . . .	123
5.6.4	Choosing Delay-Cost Parameter $V$ Autonomously . . . . .	125
5.6.5	Learning Delay Lookup Table Online . . . . .	126
5.6.6	Selecting Among Different Types of VMs. . . . .	128
5.6.7	Supporting Reserved Instances. . . . .	129
5.6.8	Sensitivity Study . . . . .	131
5.7	Summary . . . . .	132
6.	CONCLUSION . . . . .	133
6.1	Summary of the Dissertation . . . . .	133
6.2	Future Direction . . . . .	134
	BIBLIOGRAPHY . . . . .	137
	VITA . . . . .	148

## LIST OF FIGURES

FIGURE	PAGE
3.1 Electricity trace data . . . . .	32
3.2 Trace data for January 1–25, 2012. . . . .	47
3.3 Impact of $V$ . . . . .	49
3.4 Impact of time-varying $V$ . . . . .	51
3.5 Comparison with PerfectHP. . . . .	52
3.6 Comparison with PriceUn. . . . .	53
3.7 Robustness against price prediction errors. . . . .	55
3.8 Robustness against workload prediction errors. . . . .	55
3.9 Comparison between simulation and experiment results. . . . .	58
3.10 Average value versus $V$ . . . . .	60
4.1 Hybrid data center infrastructure. . . . .	63
4.2 Distributed data center model . . . . .	66
4.3 CAGE solution overview . . . . .	78
4.4 Trace data used in simulation and system experiment. . . . .	79
4.5 Location of the data centers and load balancers. . . . .	79
4.6 Comparison among benchmarks . . . . .	82
4.7 Impact of carbon-cost parameter $\sigma$ . . . . .	84
4.8 Impact of peak renewable energy . . . . .	85
4.9 Modeling 95th percentile delay and power consumption. . . . .	87
4.10 Comparison among benchmarks for system experiments . . . . .	88
5.1 Architecture diagram of BATS autoscaler. . . . .	109
5.2 Workload traces. . . . .	113
5.3 RUBiS load response time correlation. . . . .	114
5.4 Comparing BATS with other algorithms. . . . .	115

5.5	Resource allocation comparison. . . . .	116
5.6	Impact of users budget. . . . .	119
5.7	Impact of $V$ on the delay and total cost. . . . .	120
5.8	Simulator validation. . . . .	122
5.9	Average delay comparison with other algorithms. . . . .	124
5.10	Tail delay comparison with other algorithms. . . . .	124
5.11	Adaptive $V$ . . . . .	126
5.12	Delay performance with adaptive delay lookup table. . . . .	127
5.13	Impact of horizontal and vertical scaling. . . . .	128
5.14	Impact of Reserved Instances on Average delay. . . . .	129
5.15	Sensitivity study. . . . .	130

# CHAPTER 1

## INTRODUCTION

### 1.1 Motivation

Data centers are continuously growing in both numbers and sizes to meet the surging demand for online computing, increasing their electricity consumption and carbon footprints worldwide. Today's large data centers host hundreds of thousands of servers and the peak power rating of a single data center may even exceed 100MW [60]. Currently, the data centers consume 3% of global electricity production and would rank 5th in the world if the data centers were a country [44]. A significant portion of this electricity is produced from carbon-intensive sources (e.g., coal and oil), often called "brown energy". Due to the brown energy consumption, data centers are accountable for emitting 200 million metric tons of carbon dioxide per year [49, 63]. This huge amount of carbon emission has raised serious concerns about data center energy consumption and its negative impact on the environment. Governments are aggressively looking to reduce the carbon emission by introducing energy usage cap, carbon tax, and tax incentives for greener operation [7, 8]. Many IT organizations (e.g., Apple, Facebook, and Google) are consistently pressured, both from utility providers and governments, to reduce their carbon footprint and energy consumption [7, 64, 74, 94, 117]. While these companies have taken steps to reduce their carbon footprints (e.g., by installing on-site/off-site renewable energy facility), they are consistently looking for new approaches to reduce their energy consumption and carbon footprints without incurring significant additional operational costs [12, 36, 74]. Motivated by such trends, this dissertation aims to address key sustainability issues in today's data centers.

Reducing the IT energy consumption and carbon footprints of an organization mostly depends on its data center architecture, renewable energy usage, cooling efficiency, and

the carbon efficiency of the electricity generation at the data center location. The IT organizations often use two major types of data centers to cater their computing needs: self-managed and colocation data centers [12, 55]. Self-managed data centers are owned and operated by the organization and require huge initial setup cost and manpower to manage the facility. Many IT organizations need to deploy their servers in multiple geographical locations to improve service performance for local workloads. However, it may not be economically feasible for an organization to build and manage a data center at every location due to initial capital costs. In such scenarios, colocation data centers play a crucial role. In a colocation, the organization is a tenant and pays rent for space and energy consumption while the colocation operator manages the facility. Thus, to achieve low-cost global presence, the organization rents the space in a colocation data center facility and places its servers there instead of building and managing an entire data center by itself. Unlike self-managed data centers, the organization has no control over the cooling or power infrastructure of the colocation. We outline the architectural differences between a self-managed data center and colocation data center in Section 4.1. However, the colocation data center has long been ignored by the academia while there are numerous studies focusing on how to improve the sustainability for self-managed data centers [36, 65, 68, 90, 92, 93]. Motivated by the lack of sustainability study in colocation data centers, this dissertation aims to develop sustainable algorithms to reduce carbon emission in both self-managed and colocation data centers.

Integrating data center resource and workload scheduling algorithms in existing infrastructure are difficult and even impossible if they require the modification of data center architecture or existing hardware. Our motivation is to design solutions that are compatible with current data centers and applicable without the need for any physical modification. We focus more on the software-based and algorithmic aspect of the solution for their excellent feature of natural incorporability into the existing system.

Existing research for data center sustainability can be classified into two broad categories: *workload management* and *resource management*. The workload management techniques involve scheduling among geographically distributed data centers, deferring workload for later processing, and partial execution [36,68,111,112]. Resource management techniques include managing the underlying computing and infrastructure resources such as servers, physical CPUs, UPS, and generators [25,33,45,119]. In this dissertation, we study both workload management and resource management techniques. The experimental results show that our work improves carbon emission by thousands of tons and save millions of dollars by reducing the energy consumption of an IT organization.

## **1.2 Problem Definition and Contributions**

This dissertation aims to identify the key challenges and explore the sustainable resource management and workload management for cloud data centers from the following aspects:

### **1.2.1 Sustainability in a self-managed data center**

Several companies (e.g., Google and Microsoft) have declared *carbon neutrality* (a.k.a. net-zero carbon emission) as their long-term strategic goals for various reasons such as government tax incentives, public image improvement, etc. [27,41]. An organization must reduce its carbon emission level to zero over a long (e.g., six months or one year) period to claim carbon neutrality. In this part, we address how to reduce the carbon emission of a self-managed data center to “net-zero” level and achieve carbon neutrality without incurring a significant additional operational cost. The operational cost may include electricity cost, delay cost, server on/off transition cost, etc. Since carbon neutrality is a long term goal, achieving it is significantly challenging because of the unknown or intermittent nature of future workload, unit electricity price, renewable energy availability, etc. For



example, the supply of solar and wind energies heavily depend on weather conditions, and workload may increase due to breaking events. Some initial efforts have been made to achieve energy capping and carbon neutrality for data centers [64, 94, 117], but they require accurate prediction of long-term (e.g., six months or one year) future information that is generally unavailable, even impossible, to acquire in practice. Our solution achieves carbon neutrality using only past and instantaneous data, without requiring any long-term future information. Furthermore, we integrate demand responsive electricity price enabled by the emerging smart grid technology and show its benefits in reducing the data center operational cost.

**Contributions.** Our contributions to sustainable resource management in a self-managed data center are summarized below:

1. We propose an efficient online algorithm, called CNDC (optimization for Carbon-Neutral Data Center), to control the number of active servers for minimizing the data center operational cost under carbon neutrality. To our best knowledge, CNDC takes the first step towards carbon neutrality while incorporating demand-responsive electricity prices as well as multiple data center energy sources (e.g., electricity, on/off-site renewable energy, and RECs).
2. We leverage the existing Lyapunov optimization technique which dynamically adjusts the tradeoff between cost minimization and the potential deviation from carbon neutrality. We formally prove that CNDC is efficient and provides an analytical performance bound compared to the optimal offline algorithm that has future information.
3. We also perform trace-based simulation and experiment studies to complement the analysis. The results show that CNDC reduces cost by more than 20% while achieving lower carbon footprint in an online manner, compared to prediction-based meth-

ods. Compared to taking the electricity price as is without incorporating demand responses, CNDC further decreases the average cost by approximately 2.5%, translating into hundreds of thousands of dollars per year.

### **1.2.2 Sustainability in a hybrid data center infrastructure**

Besides self-managed data centers, many IT organizations often lease space and house their servers in geo-distributed colocation data centers, where they share the power (including renewables) with other tenants. Thus, many organizations use a hybrid data center infrastructure, where some computing servers are managed in-house (a.k.a. self-managed) data centers, and the rest are placed in colocations [62]. Sharing of renewable energy at a colocation creates new challenges: how can an organization minimize its carbon footprint in colocations? While numerous studies have investigated geographical load balancing to minimize carbon emissions of data centers, these studies have primarily focused on self-managed data centers where all the renewables are solely dedicated to the data center operator. Furthermore, colocation data centers have different cost and operational structure for energy usage. In this part, we consider a practical hybrid data center infrastructure (including both self-managed and colocation data centers) and propose a novel workload and resource management algorithm based on alternating direction method of multipliers to reduce carbon footprints. Our solution dynamically distributes incoming workloads to geo-distributed data centers based on local renewable availability, carbon efficiency, electricity price, and also the energy usage of other tenants that share the colocation data centers.

**Contributions.** Our contributions to sustainable workload and resource management in a hybrid data center infrastructure are summarized below:

1. We identify the problem and challenges of minimizing the carbon emission in a hybrid data center infrastructure which is very common in practice for supporting large organizations' computing needs.
2. We propose an efficient online algorithm, called CAGE (**C**arbon and **C**ost **A**ware **G**Eographical Job Scheduling), to control the number of active servers for minimizing the data center operational cost under carbon neutrality. To our best knowledge, CAGE takes the first step towards sustainable resource management in a hybrid data center infrastructure.
3. We propose a distributed workload and resource management algorithm, based on the *alternating direction method of multipliers* (ADMM), that optimizes GLB decision for a hybrid data center to minimize carbon emission, electricity cost, and revenue loss while meeting performance requirements. CAGE leverages the geographical variation of carbon usage effectiveness (CUE) of electricity production, renewable energy, electricity cost, and cooling efficiency for each data center location. The optimality of our solution can be guaranteed under mild practical assumptions.
4. We perform extensive trace-based simulation and real life system experimental studies to show the effectiveness of our solution. Our study shows that CAGE can reduce carbon emission by 6.35%, 9.4%, and 37% compared to the algorithm that ignores colocation data centers, minimizes cost, and maximizes performance, respectively.

### **1.2.3 Sustainability from a cloud service user's point of view**

Many companies (e.g., Microsoft and Amazon) provide data center resources in the form of cloud services to the users. Cloud services have evolved into diversified models, such

as Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), which relieves users from the hassle of maintaining their own infrastructure and cater to a wide spectrum of user needs such as scientific computing and web hosting. Generally, a user manages the purchased resources (e.g., virtual machines) and has no direct control over the corresponding data center energy consumption. An intuitive approach to limit the energy consumption or carbon emission from a user’s side is to limit the number of purchased resources. Furthermore, a recent survey [103] shows that 80% small and medium sized cloud service users are given a specific amount of budget by business departments or higher-level executives at the beginning of a budgeting period (e.g., typically, a month or a year). Such budget constraints are also commonly applied to universities and governments, which typically allocate annual IT operational budgets at the beginning of each fiscal year [105]. Thus, setting a user spending limit or budget for cloud services is a win-win scenario: addressing both economical and environmental sustainability issues for the user. In the final part of this dissertation, we address how to optimize the delay performance of a cloud service/application while meeting long-term budget constraints using only past and instantaneous workload information.

**Contributions.** Our contributions are summarized below:

1. We develop an online autoscaling system, called BATS (**B**udget-constrained **AuT**O-scaling), that dynamically scales VM instances to optimize the delay performance while satisfying a user’s budget constraint in the long run. Leveraging Lyapunov optimization technique, we formally prove that the BATS produces a close-to-optimal delay performance compared to the optimal algorithm with offline information while satisfying the budget constraint.
2. As a system, we build a *fully-automated* BATS autoscaler service on Windows Azure. BATS autoscaler only requires user inputs on the desired delay performance and budget of their applications. It manages the performance monitoring, resource

planning, and scaling of user applications automatically. We also combine BATS algorithm with a reactive module that monitors runtime performance and handles workload burstiness.

3. We also conduct extensive simulation study that complements the system implementation results. We evaluate BATS in terms of both average delay and 95<sup>th</sup> percentile delay, showing the effectiveness of our algorithms on different performance metrics and its scalability on managing applications with hundreds of VMs. We also show that BATS is truly autonomous; it does not need users to select appropriate algorithm parameters or provide additional application information.

### 1.3 Related Publications

This dissertation work is drawn from the following publications:

- A. Hasan Mahmud and Shaolei Ren. Online capacity provisioning for carbon-neutral data center with demand-responsive electricity prices. *ACM SIGMETRICS Performance Evaluation Review*, 41(2):26–37, 2013.
- A. Hasan Mahmud and S. S. Iyengar. A distributed framework for carbon and cost aware geographical job scheduling in a hybrid data center infrastructure. In *The 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016.
- A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. BATS: budget-constrained autoscaling for cloud performance optimization. In *IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MAS-COTS)*, 2015.

## 1.4 Outline of the Dissertation

We discuss related work in Chapter 2. We first provide a snapshot of the current data center resource management techniques (that we leverage) in Section 2.1, 2.2, 2.3, and 2.4. Then, we describe the key difference between our work and the existing studies in Section 2.5.

We present how to address *sustainability in a self-managed data center* in Chapter 3. Section 3.1 provides background information and challenges. Section 3.2 discusses modeling details, and Section 3.3 presents the offline problem formulation and develops an online algorithm, CNDC, which explicitly incorporates demand-responsive electricity prices. Section 3.4 and 3.5 show the simulation and system experimental results, respectively.

Chapter 4 describes our approach to address the issue of *sustainability in a hybrid data center infrastructure*. Section 4.1 discusses background information and challenges. Section 4.2 models a hybrid data center infrastructure. Our distributed solution is presented in Section 4.3, followed by simulation and experimental results in Section 4.4 and 4.5, respectively.

Chapter 5 develops a solution for cloud service users to cap their spending on the cloud resources to address *sustainability from a cloud service user's point of view*. Section 5.2 presents the model and problem formulation. In Section 5.3, we develop our online autoscaling algorithm, BATS. Section 5.4 describes the system implementation for Microsoft Azure. Sections 5.5 and 5.6 provide real-world experimental and simulation studies, respectively. Finally, we provide our concluding remarks and future direction in Chapter 6.

## CHAPTER 2

### RELATED WORK

This dissertation leverages recent mechanisms in data center sustainability research: dynamic server provisioning, geographical load balancing, colocation data center resource optimization, and autoscaling of virtualized resources. First, we present related work for each of these mechanisms and then discuss the difference between our work and existing studies.

#### 2.1 Dynamic Server Provisioning

All the solutions we proposed in this dissertation use dynamic server provisioning. The key idea in dynamic server provisioning is to turn off or lower the CPU frequency of as many servers as possible while maintaining the desired service performance. It has been extensively used to minimize the energy consumption and operational cost of a self-managed data center. For example, [33] uses a queueing theoretic model which predicts the mean response time for a data center as a function of some input variables such as power-to-frequency relationship, arrival rate, peak power budget, etc. Using this model, their algorithm determines the optimal power allocation for every possible configuration of the input variables. [66] addresses how to determine the number of servers required to serve the incoming workloads. They propose an online solution that predicts the future arrivals and uses dynamic server provisioning to turn the servers on/off to minimize operational costs. The performance of their analytical solution is bounded to that of offline optimal.

Among other work, [45] considers three key trade-offs of data center resource management that uses dynamic server provisioning. Firstly, it addresses maximizing energy savings vs. minimizing unmet load demand. It argues that aggressive server provisioning

may degrade the application performance while saving energy. Thus, the provisioning algorithm must maintain specified Service-Level Objectives (SLOs). Secondly, it considers putting a server in low power state (using DVFS) rather than completely turning it off to avoid latency between off-on switching. Finally, it considers the trade-off between energy savings and reliability costs of servers due to on-off cycles. Aggressive server provisioning (repeated on-off cycles) may lead to the failure of server components (e.g., hard drive), incurring additional costs for their replacement. Hence, the solution considers server on-off latency, SLO requirements, and server reliability costs while making dynamic server provisioning decision. Furthermore, [112] combines dynamic server on-off and geographical load balancing to reduce energy consumption and electricity cost while maintaining satisfactory service performance.

Considering the power outages in data centers, [110] reduces the energy cost of geographically distributed data centers while guaranteeing the performance requirement of dynamic workload. Their proposed solution leverages dynamic server provisioning and load shifting among data centers. The optimum solution of the problem is based on the stochastic multiplexing gain in a cross-data center level which yields slightly better results compared to the previous approach.

**Dynamic Voltage and Frequency Scaling (DVFS).** It has been used widely to reduce active server power by lowering the operating voltage and frequency of the processor. In contrast to server on-off latency, switching in and out of DVFS states are very fast, typically in the order of tens of a microsecond [59]. Hsu and Feng propose an algorithm using DVFS that reduces the energy consumption while keeping the performance degradation bounded [48]. Using an estimation model that relates the off-chip access to CPU usage intensity, their solution computes the lowest frequency and voltage required for CPU such that the server performance is not significantly affected. [14] shows that DVFS can reduce server power consumption when servers are operating in a typical range of 10% to 50%



utilization. [11] explores power consumption reduction by transitioning to the C (sleep) states whenever the CPU is idle. The deeper C-states turn off L1 and L2 caches. The cache contents need reloading each time transitioning from a C-state, which hurts performance if CPU is switching back and forth from C-states frequently. The proposed solution (Idle-Power) consolidates the CPU timing interrupts and increases the deep C-state residency of the processor to reduce CPU power consumption. Deng et al. [25] show that DVFS can be applied to other modern server components, such as DRAM devices and memory controller, at the time of low usage to reduce energy consumption. While these studies are encouraging, DVFS only applies to the CPU and some other components of a server, significantly limiting its potential power savings capability.

## **2.2 Geographical Load Balancing (GLB)**

Our proposed solution for addressing sustainability in a hybrid data center infrastructure uses geographical load balancing (GLB). To achieve low-cost global presence and to improve service performance, many organizations are using data centers that are geographically distributed all over the world. Multiple data centers present a unique opportunity for the organization where it can use GLB to leverage geographical diversities of electricity cost, renewable energy availability, efficiency, etc., and distribute the workload to different data center locations to minimize the objective function (e.g., carbon footprint). There have been a significant number of prior studies that considers GLB for data center workload scheduling. For example, [92] develops a GLB algorithm by considering the location varying electricity price to reduce the electricity bill of an organization that has multiple data centers in geographically distributed locations. In most parts of the U.S. with wholesale electricity markets, unit prices may vary (e.g., 15 minutes, 30 minutes, or on an hourly basis) as much as a factor of 10. Their solution leverages the fact that electricity prices are not well correlated at different locations. During low computational

demand, their algorithm routes the workload to a data center that has low electricity price and reduces the total electricity cost for that organization without hurting the service performance significantly.

[93] considers a multi-electricity-market environment and then formulate the electricity minimization problem as a constrained mixed-integer programming problem to minimize the total electricity cost. Compared to the previous work, the notable differences of this approach are: (1) it captures the tolerable service delay requirements using a constraint rather than minimizing it, and (2) it solves the linear programming formulation with Brenner's fast polynomial-time method that yields better results [19].

[36] leverages multi-location varying carbon efficiency of electricity production and electricity price to reduce the carbon footprint and cost of geographically dispersed data centers. Their proposed solution FORTE, a flow optimization based framework for request-routing, uses an objective function that balances the weighted sum of access latency, electricity costs, and carbon footprint of geographically distributed data centers. Using this three-way trade-off, FORTE analyzes the costs of carbon emission reduction for a large-scale Internet application. They consider Akamai content distribution network (CDN) for their case study. Based on the analysis, authors also discuss whether it is beneficial for an operator to use (i.e., being upgraded to the green data center) FORTE.

[68] develops a distributed algorithm for geographical load balancing with provable optimality guarantees and explores the feasibility of using GLB for demand response in the grid environment. In their provable optimal solution, each data center minimizes the cost, which is a linear function of energy cost and delay cost. The delay cost is measured as the lost revenue due to the delay of requests which includes both network propagation delay and workload intensity dependent queueing delay at the server end. The geographical load balancing algorithm decides the routing of the requests to a data center, and the number of servers required in the on-state (active) to serve the incoming workload. The

authors further explore the feasibility and benefits of using GLB to incorporate the renewable sources into the grid. They consider time-varying and location-varying renewable (e.g., wind and solar) energy availability and combines both demand response program and dynamic electricity pricing. Authors show that when the data center incentive (from demand response program) is aligned with the goal of brown energy consumption reduction, their approach can provide significant social benefit such as reducing the brown energy consumption or reducing the carbon emission.

[112] exploits the temporal and spatial variations of workload arrival and electricity prices to reduce energy consumption cost. The temporal job scheduling makes their algorithm more suitable for the delay tolerant batch workloads such as MapReduce jobs. Unlike delay sensitive interactive jobs, the execution of the delay tolerant batch workload can be deferred for a long (e.g., hours) time. Its two-time scale (temporal and spatial) control algorithm reduces electricity cost by considering electricity cost vs. delay trade-off in geographically distributed data centers. The authors show that by extending (from a single time scale to two different time scales) the Lyapunov optimization approach, the average power cost and service delay achieved by the algorithm can be analytically bounded.

[116] maximizes renewable energy usage in geographically distributed data centers through dynamic load dispatch, subject to the monthly budget determined by the Internet service operator. This study models renewable energy generation (e.g., solar panels), with respect to the varying weather conditions of each data center locations, to handle the intermittent supplies of renewable energy. Finally, the authors transform their optimization problem into a linear-fractional programming (LFP) formulation for a GLB solution with a polynomial time average complexity.

By considering the smart micro grids and frequent power outages in a data center, [114] formulates a stochastic program to minimize the long-term energy cost of dis-

tributed Internet data centers while capturing the service request distribution, server provisioning, battery management, generator scheduling, power transactions between smart micro grids and main grids. The authors argue that, if power outages occur frequently (e.g., due to cyber attacks) in a data center which is operating in a smart grid environment, the cumulative cost of running the diesel generators will become very high, and hence, a smart management is required for the UPS, generators, workloads, etc. Their solution leverages Lyapunov optimization technique and provides a performance guarantee (theoretically) for their algorithm by enabling an explicit tradeoff between energy cost saving and battery investment cost.

[118] proposes dynamic pricing of VM resources in geographically distributed data centers to maximize the long-term profit of the cloud provider. The authors argue that the resource cost of each location is different and time varying depending on the other factors such as electricity price, etc. Thus, VM pricing at each location should be adjusted by the cloud service provider to maximize the profit. Authors leverage dynamic server provisioning and Lyapunov optimization technique to propose an online solution that guides the operational decisions of the cloud provider to pursue maximum averaged long term profit.

### **2.3 Colocation Data Center**

We incorporate colocation data centers while addressing sustainability in a hybrid data center infrastructure. There are some recent studies on reducing the energy consumption and carbon emission of the colocation data centers. In a colocation environment, multiple tenants house their server in a facility which is owned by a third party. Unlike self-managed data centers, the tenants at a colocation data center have no direct control over the cooling or facility management. Thus, reducing the energy consumption or carbon emission requires coordination among tenants and the colocation operator. Most coloca-

tion data center resource management algorithms leverage either demand response (DR) or emergency demand response (EDR) program provided by the smart grid environment. Hence, we briefly discuss these techniques. Demand response provides an opportunity for consumers to lower their electricity bills by reducing or shifting their electricity usage during peak periods in response to time-based rates where the electricity rate is higher during the peak hours [29]. For example, a household consumer can save money by using their washer and dryer during off-peak hours when the electricity rate is cheaper. In emergency demand response (EDR) program, the utility provider gives financial incentives to the businesses that reduce their electricity consumption upon receiving a signal. EDR allows the utility provider to keep the electricity demand within the supply capacity, and hence, prevent blackouts.

[96] discusses the challenges of colocation data center resource management. The main difficulty in a colocation data center is that it has no control over the servers placed by the tenants. Hence, it is not possible for the colocation data center operator to reduce the energy consumption by turning off the servers like self-managed data centers. The authors argue that the colocation data center operator and tenants need a collaboration mechanism. The authors propose an algorithm to reduce the energy consumption of a colocation data center by using a reward based bidding mechanism for participating tenants. Their work assumes that the colocation operator is participating in the emergency demand response (EDR) program. The tenants submit bids indicating how much energy consumption they want to reduce and its corresponding reward amount. The colocation operator accepts suitable bids to minimize total energy consumption, and the total reward provided to the tenants does not exceed the financial incentives received from the utility provider. If a bid is selected, the corresponding tenant reduces its energy consumption and receives financial reward from the colocation operator. Participation of a tenant is voluntary, making it suitable for any colocation data center. [54] extends the studies of [96] by

considering equivalent carbon emission reduction instead of energy consumption while selecting bids.

By coordinating the participating tenants in a colocation data center, [55] reduces the electricity cost by lowering the energy consumption from colocation operator's perspective. Their solution considers time-varying data center cooling efficiency, assuming that the data center cooling system uses air-side economizer which cools the data center using cold outside air whenever applicable. It also considers peak power demand charge which may take up to 40% of colocation operators total electricity bill. The peak power demand charge is not known perfectly until the end of a billing cycle. This solution uses a feedback-based online optimization by dynamically keeping track of the maximum power demand during runtime to compensate peak power demand charge.

[102] proposes an auction-based fair reward(incentive) system for the participating tenants in a colocation data center to reduce cost and carbon emission by leveraging EDR. Their proposed solution focuses on the auction mechanism, called FairDR, which simplifies it. Typically, when random energy reduction signals from utility provider arrive at a colocation data center, participating tenants have to submit bids for energy reduction each time. In sharp contrast, FairDR collects the bidding information from tenants only once, while tenants' actual energy reduction is decided online for subsequent EDR signals. Furthermore, FairDR guarantees that similar rewards are offered to tenants with similar energy reduction.

## **2.4 Autoscaling of Virtualized Resources**

We leverage autoscaling of virtualized resources while addressing sustainability from a cloud service user's point of view. In general, autoscaling techniques can be classified into two broad categories: *proactive* and *reactive*. In *reactive* autoscaling, decisions are actively triggered by a user at the beginning of a decision period via, e.g., predictive

modeling of the workload [21, 32, 40, 98, 100]. In contrast, *reactive* autoscaling decisions are made in passive response to a system's current statuses (e.g., CPU, memory, and I/O utilization) [79, 83]. Below, we present a snapshot of related work for both of these approaches.

#### 2.4.1 Proactive autoscaling

[40] introduces PRESS, a statistical learning algorithm to predict CPU usage of virtual machines in an online fashion. It derives a signature for the pattern of historic resource usage and uses that signature in its prediction. This *signature* driven prediction works better for the workloads with repeating patterns. It also uses state driven prediction to improve the accuracy for other types of workload. [100] discusses CloudScale, a set of schemes to improve the prediction-driven resource scaling system by using error correction mechanisms. For under-estimation error correction, CloudScale uses online adaptive padding and reactive error correction. The authors argue that reactive error correction alone is often insufficient. When an underestimation error is detected, SLO violation has already happened. A proactive padding is added to avoid such violations. The algorithm chooses the padding value based on the recent burstiness of application resource usage and recent prediction errors.

[98] develops a model predictive algorithm for workload forecasting and uses it for resource autoscaling. It uses a second order autoregressive moving average method (ARMA) for workload prediction. [21] uses autoregression techniques to predict the request rate for a seasonal arrival pattern, and then accordingly turns servers on and off using a simple threshold policy. The dynamic provisioning policy performs well for periodic request rate patterns that repeat, say, on a daily basis.

[32] explores a new cloud service, Dependable Compute Cloud (DC2), that automatically scales the infrastructure (number of VM for application layer) to meet the user-

specified performance requirements (e.g., SLAs). DC2 is application-agnostic and does not require any offline application profiling or benchmarking. This service allows a CSP (Cloud Service Provider) to resize user applications to meet performance requirements in a cost-effective manner. In general, CSPs cannot gather all the necessary application-level statistics (e.g., response time) without intruding into user-space application. DC2 address this challenge by leveraging Kalman filtering to automatically learn the system parameters for each application and proactively scales the infrastructure. When executing, the change in the workload causes a change in the service time of the requests. The Kalman filter detects this change based on the monitored values and estimates the new system state which indicates the required scaling actions for DC2.

#### **2.4.2 Reactive autoscaling**

[79] presents an autoscaling solution to minimize job turnaround time within budget constraints for cloud workflows. This simulation based study is limited to only cloud workflows. Every workflow job has a priority and is composed of connected tasks in the form of a directed acyclic graph (DAG). This study presents two algorithms: scheduling-first and scaling-first. Scheduling-first algorithm first allocates the service provider budget to each job based on the job priority and then schedules as many tasks as possible to their fastest execution VMs within job budget constraint. When the VM type for each task is determined, the autoscaling mechanism acquires VM instances based on the scheduling plan. The scaling-first algorithm first determines the type and the number of cloud VMs within the budget constraint and then, schedules jobs on the acquired resources based on job priority to minimize weighted average job turnaround time. The scaling-first algorithm first makes resource scaling decisions and then makes job scheduling decisions. The scaling-first algorithm shows better performance when the budget is low while the scheduling-first algorithm performs better when the budget is high.



[34] introduces a dynamic capacity management policy, called AutoScale, which reduces the number of servers needed in data centers driven by time-varying load while meeting response time SLAs (service level agreements). The goal is to minimize energy consumption while meeting SLA constraints in a reactive manner. This solution is conservative while scaling down the number of servers by not turning servers off recklessly. It first determines the service rate (Reqs/sec) of a server when meeting 95<sup>th</sup> percentile delay of 400 ms. Then, it keeps an additional capacity margin of 100% when scaling up and scales down only if servers are idle for a specific period.

[83] discusses the performance improvement and cost reduction of a real production environment (in AWS) by using AutoScaling. Their study considers a cloud-hosted application that provides sports fans with real-time scores, news, photos, statistics, live radio, streaming video, etc., on their mobile devices. Spikes in the workload happen very quickly and mostly during a game time. The scaling algorithm follows an aggressive scale-up policy to cope with spiky workloads but scales down very conservatively. It downscales by removing only one VM at a time and making sure that the CPU usage of content-delivery tier is stable for a long period of time before doing another round of downscaling. The number of users may suddenly decrease during half time of the game, and if the algorithm is not conservative in downscaling, servers may overload when half-time is over.

## **2.5 How Our Work is Different?**

We now discuss the difference between existing studies and this dissertation.

### **2.5.1 Sustainability in a self-managed data center**

In the first part of this dissertation, we address how to reduce the carbon emission of a self-managed data center to “net-zero” level and achieve carbon neutrality using only in-

stantaneous and past information. Most studies provide solution for short-term (e.g., an hour) carbon emission reduction, and these solutions are not applicable for the long-term carbon neutrality problem that we are considering [36, 67, 68, 116]. We are addressing how to reduce carbon emission for many short-term decision period such that the net carbon emission is zero after a long budgeting period (e.g., six months or one year). Existing studies to reduce and cap carbon emission rely on long-term prediction of the future information (e.g., renewable energy availability, workload demand, and demand responsive electricity price), which may not be accurate and in many cases are impossible to acquire in practice [24, 64, 94] In sharp contrast, our solution does not require the prediction of long-term future information. Furthermore, several heuristic algorithms have been proposed by keeping additional resource allocation margins to compensate the uncertainty in workload prediction [24, 64], their evaluation is based on the empirical results and do not provide any analytical performance guarantees. Our prior work [76, 77] proposes online capacity provisioning algorithms for energy capping without incorporating demand-responsive electricity price. In summary, our solution, called CNDC, offers provable guarantees on the average cost, theoretically bounds the deviation from long-term carbon neutrality, and considers emerging demand-responsive electricity price. Our simulation results also demonstrate the benefits of CNDC over the existing methods empirically.

### **2.5.2 Sustainability in a hybrid data center infrastructure**

There have been numerous studies on optimizing power management in data centers. For example, [34, 45, 66] use dynamic server provisioning to balance between energy consumption and application performance. Minimizing energy consumption and carbon emission via geographical load balancing for self-managed data centers have been extensively studied. Using GLB for distributed data centers, [23, 36, 68] exploit the spatio-temporal diversity of carbon efficiency and renewable energy availability to reduce carbon

footprints. [90, 92, 93, 120] minimize power cost by considering location varying electricity prices among multiple geographically distributed data centers. Capping the carbon emission and energy consumption of a self-managed data center have been studied in [74, 76, 116]. Using ADMM for geo-distributed data centers, [111] reduces electricity demand charge by partially executing search queries and [70] minimizes the energy consumption considering heterogeneous servers in data centers. These studies, albeit promising, ignore the power sharing in colocations among multiple tenants, and hence, are not applicable for a hybrid data center infrastructure that are very common in practice.

Some recent studies [55, 58, 96] have investigated market approaches to coordinate tenants' power consumption for reducing electricity cost and carbon emissions in the context of colocations. For example, tenants are paid rewards to shed energy consumption to avoid simultaneously peaking their power usage to reduce peak demand charge [55]. In other study, tenants engage in power reduction through supply function bidding to tackle power emergencies caused by oversubscription [58]. [121] proposes an incentive-based solution by leveraging ADMM to enable demand response for geo-distributed colocation data centers. These studies, however, are focused on the colocation operator's pricing decision. In sharp contrast, our study focuses on the tenant/organization and proposes to optimize the organization's GLB in a hybrid data center infrastructure. To our best knowledge, this dissertation is the first to consider the unique sharing of power infrastructure (and hence, renewables, too) in colocations for optimizing a tenant's GLB decisions.

### **2.5.3 Sustainability from a cloud service user's point of view**

We leverage autoscaling of virtualized resources to address this issue. In recent years, autoscaling has become an integral feature of cloud computing, and various autoscaling mechanisms have been proposed to enable elastic resource acquisitions for performance and cost effectiveness. We discuss the well-known autoscaling algorithms in Sec-

tion 2.4. Existing studies use both *proactive* and *reactive* autoscaling. For example, [21, 40, 98, 100] use prediction/learning techniques to estimate workload demand/arrival rates for autoscaling, while [32] builds a performance model to make autoscaling decisions. Many cloud service providers offer both schedule-based and rule-based “reactive” autoscaling [1, 4–6, 9]: cloud users can specify customized schedules to initiate/release VM instances at particular times using schedule-based autoscaling while rule-based autoscaling scales VM instances based on resource usage thresholds (e.g., CPU, memory usage). Nonetheless, as pointed by [83], these autoscaling rules often result in unnecessarily high costs, because they are difficult to be optimally tuned toward a long-term budget goal because of highly dynamic workloads and over-subscription of VM resources for additional capacity headroom. On the contrary, our proposed solution, called BATS, guides the spending for VM subscription to satisfy long-term budget constraint while it exploits benefits of both proactive and reactive scaling. Specifically, we scale resources proactively based on short-term workload prediction, while we also incorporate reactive autoscaling as a backup during exceptions (e.g., workload spikes).

There have been some prior studies on satisfying *short-term* budget constraint. For example, [80] uses a constant hourly budget to decide the optimal number of VM instances for jobs that have a larger deadline (e.g., 1 hour), while [79] also scales and schedules cloud workflows considering the hourly budget constraint for each individual job. Similarly, [78, 99, 113] optimizes workflow scheduling by exploiting flow-specific properties (e.g., user-specified priorities) while considering an instantaneous budget constraint. These studies only impose a short-term (e.g., hourly) budget constraint, which bounds resource availability at each step independently. Our work considers an even harder problem: the resource availability over different time steps is dependent as we bound the total resources across the entire budgeting period (e.g., weeks). Moreover, they focus on delay-tolerant scientific/batch (e.g., large-scale simulation, and video pro-

cessing) jobs, whereas we focus on delay-sensitive (e.g., web services) interactive jobs. Scheduling delay-tolerant jobs are easier under a short-term budget constraint because their processing can be deferred for later decision periods whereas interactive jobs must be processed as they arrive.

Our work broadly lies in the category of dynamic resource management and hence, is also related to some other domains, such as server management in data centers [26, 34]. While many efforts have been dedicated to enable autonomic and self-managing systems (using control theoretic and learning approaches) [26, 87, 101], only a few address long-term performance/constraints. For example, the existing research that deals with long-term constraints (e.g., brown energy [64], monthly cost [117]) in data centers often relies on accurate predictions of future information that may not be available in practice. To the best of our knowledge, we develop the first provably-efficient online autoscaling solution to optimize delay performance for real-world cloud applications while satisfying a long-term budget constraint.

To reduce the cloud service cost, recent studies propose hybrid VM rental decisions: dynamically request on-demand instances to cope with workload variations, while using reserved instances to serve base workloads for cost saving [108]. Reserving VM instances incurs high upfront reservation fees and often requires yearly or even longer commitment in practice (e.g., Amazon EC2 [1]). Our study also explores incorporating reserved instances to provide cost savings.

## CHAPTER 3

### SUSTAINABILITY IN A SELF-MANAGED DATA CENTER

Driven by the exploding demand for Internet services, data centers are growing continuously in both number and scale, resulting in a huge amount of energy consumption. Recent studies have shown that the combined electricity consumption of global data centers amounts to 623 billion kWh annually and would rank 5th in the world if the data centers were a country [44]. As a significant portion of electricity is produced by coal or other carbon-intensive sources, it is often labeled as “brown energy” and the growing trend of data center electricity consumption has raised serious concerns about the sustainability. In this part of the dissertation, we discuss a novel algorithm to reduce and cap the carbon emission of a self-managed data center to “net-zero” level.

#### 3.1 Background

Recently, several companies such as Google and Microsoft have declared *carbon neutrality* (a.k.a. net-zero) as their long-term strategic goals: reducing the net carbon footprint to zero [27,41]. A widely-used approach to accomplishing this goal is “first reduce what you can, then offset the remainder”. While various power management techniques (e.g., power proportionality via turning on/off servers [34]) and integration of on-site green energy (often generated by solar panels and wind turbines) have been proposed to decrease the carbon footprint of data centers. However, brown energy (or electricity) still accounts for more than 70% of the energy portfolio in many large data centers [27,41], because the best location for building a data center may not be the most desired location for generating sufficient green energies to satisfy the data center requirement. Thus, data centers are also aggressively seeking off-site renewable energy (procured through power purchasing agreement or PPA) as well as purchasing renewable energy credits (RECs) to offset the

brown energy usage, thereby achieving carbon neutrality [27, 41]. Although completely offsetting brown energy usage with renewable energy is appealing, the operational cost of a data center (of which electricity cost is a major factor) cannot be significantly increased for economic concerns.

Achieving carbon neutrality is significantly challenging in practice: a data center needs to decide its energy usage in an online manner that cannot possibly foresee the far future time-varying workloads or intermittent green energy availability. Meanwhile, operational energy cost and quality of service (QoS) are two primary concerns of data center operations, which must not be significantly compromised by the quest for energy capping or carbon neutrality. Some initial efforts have been made to achieve energy capping and carbon neutrality for data centers [64, 94, 117]. However, they require accurate prediction of long-term future information (e.g., workloads, renewable energy availability) that is typically unavailable in practice due to unpredicted traffic spikes and the intermittent supply of solar and wind energies heavily depending on weather conditions. Thus, the lack of accurate long-term future information suggests the use of online algorithms that can be applied based on the currently available information.

With the deregulation of electricity markets, electricity prices in many regions vary over time (e.g., every hour or 15 minutes), as determined by local utility companies based on supply-demand curves [107]. Recently, leveraging the technological advancement of two-way communications, smart grid has enabled an automated management and distribution of electricity to balance the demand and supply intelligently. A unique characteristic of the smart grid is *demand response*: customers adjust their energy consumption in response to real-time pricing signals, while utility companies set their prices based on the aggregated demand. Thus, the electricity price is also called demand-responsive electricity price [107]. While individual households or other small energy consumers do not have the capability of altering the electricity price, the power consumption of a large-scale

data center is often so significant that its impact on the real-time electricity price cannot be neglected. While spatial-temporal-diversities of electricity prices have been extensively explored, most of the existing research on data center cost minimization optimizes server capacity provisioning decisions by taking the real-time electricity price *as is* without considering the impact of data center operation on electricity prices [64, 68, 93]. Recent work [107, 117] has shown that using geographical load balancing while considering *demand-responsive* electricity prices that may be affected by load dispatching decisions can decrease the electricity cost by approximately 5% in large data centers. Nonetheless, it remains unclear whether and by how much demand-responsive electricity prices contribute to electricity cost saving for a data center with a long-term carbon-neutrality constraint.

In this part of the dissertation, we propose an efficient online algorithm, called CNDC (optimization for Carbon-Neutral Data Center), to control the number of active servers for minimizing the data center operational cost under carbon neutrality. While carbon neutrality is clearly a long-term goal, CNDC does not require the far future information (which is often difficult to obtain). To our best knowledge, CNDC takes the first step towards carbon neutrality while explicitly incorporating demand-responsive electricity prices as well as multiple data center energy sources (e.g., electricity, on-/off-site renewable energy, and RECs). Leveraging and also extending the recently-developed Lyapunov optimization technique [87], we conduct a rigorous performance analysis and prove that CNDC can achieve a parameterized operational cost (incorporating both electricity and delay costs), which is arbitrarily close to the minimum cost achieved by the optimal algorithm with lookahead information, while bounding the maximum carbon deficit for almost any workload and energy availability trajectories. We also perform trace-based simulation and experiment studies to complement the analysis. The results are consistent with our theoretical analysis and show that CNDC reduces cost by more than 20% while achieving



a less carbon footprint in an online manner, compared to prediction-based methods. Compared to taking the electricity price as is without incorporating demand responses, CNDC further decreases the average cost by approximately 2.5%, translating into hundreds of thousands of dollars per year.

## 3.2 Model

In this section, we present the modeling details for workloads, data center, electricity price and renewable energy. We consider a discrete-time model by dividing the entire budgeting period (e.g., 6 months or a year) into  $K$  time slots. Each time slot has a duration that matches the timescale of prediction window for which the data center operator can *accurately* predict the future information (including the workload arrival rate, renewable energy supply, and electricity price). In the following analysis, we mainly focus on hour-ahead prediction for the convenience of presentation, while noting that the model applies as well to longer-term prediction. For example, if the operator can only predict the hour-ahead future information, then each time slot corresponds to one hour and the operator can update its resource management decisions at the beginning of each hour. Nevertheless, if the operator is able to perform a longer-term prediction (e.g., day-ahead prediction), then each time slot corresponds to the prediction window, at the beginning of which the operator needs to select a sequence of decisions that will be used throughout the prediction window. Throughout the chapter, we also use *environment* to collectively refer to electricity price, on-site/off-site renewable energy supplies and workloads. Key Notations are summarized in Table 3.1.

### 3.2.1 Workloads

We consider  $J$  types of workloads (or jobs, as interchangeably used in the chapter). Different types of jobs differ in terms of the relative importance in the total cost function as

Table 3.1: List of key Notations for CNDC Model.

Notation	Description
$\lambda_j(t)$	Arrival rate of type- $j$ jobs
$m_j(t)$	Number of servers for type- $j$ jobs
$p(t)$	Power consumption
$r(t)$	On-site renewable energy
$f(t)$	Off-site renewable energy
$u(t)$	Electricity price
$e(t)$	Electricity cost
$d(t)$	Delay cost
$g(t)$	Total cost
$V$	Cost-carbon parameter
$q(t)$	Carbon deficit queue length

well as their service rates (i.e. a server may process one type of jobs faster than another type). We denote by  $\lambda_j(t) \in [0, \lambda_{\max,j}]$  the arrival rate of type- $j$  jobs and by  $\mu_j$  the service rate of a server for type- $j$  jobs. We assume that  $\lambda_j(t)$  is accurately available at the beginning of each time slot  $t$ , as widely considered in prior work [23, 64, 68], while simulation studies show that CNDC is robust against workload prediction errors. In our study, we focus on delay-sensitive interactive jobs (as in [68, 93]), whereas delay-tolerant batch jobs can be easily maintained by a separate batch job queue as in [46, 97, 112].

To quantify the overall delay performance, we denote *delay cost* for type- $j$  jobs by  $d_j(\lambda_j, m_j)$ , which is intuitively increasing in  $\lambda_j$  and decreasing in  $m_j$  where  $m_j$  is the number of (homogeneous) servers allocated to type- $j$  jobs [68]. As a concrete example, we model the service process at each server as an M/M/1 queue, which provide a reasonable approximation for the actual service process [33, 68, 93], and use the average response time (multiplied by the arrival rate) to represent the delay cost. Specifically, the total delay cost at time  $t$  can be written as  $d(\lambda(t), \mathbf{m}(t)) = \sum_{j=1}^J w_j \cdot \frac{\lambda_j(t)}{\mu_j(t) - \frac{\lambda_j(t)}{m_j(t)}}$ , where  $w_j \geq 0$  is the weight indicating the relative importance of type- $j$  jobs (i.e., a larger weight means the delay performance is more important),  $\lambda(t) = (\lambda_1(t), \dots, \lambda_J(t))$ , and  $\mathbf{m}(t) = (m_1(t), \dots, m_J(t))$ . Note that we ignore the network delay cost, which can be

approximately modeled as a certain constant [68] and added into the delay cost without affecting our approach of analysis. In addition, our analysis is not restricted to any specific delay cost, and we will also use the actual delay measured on a real system to calculate the cost.

### 3.2.2 Data center

As in prior work [66], we consider a data center with  $M$  homogeneous servers, while noting that heterogeneous servers can also be easily incorporated. The service rate of a server for processing type- $j$  jobs is  $\mu_j$  (quantified in terms of the average number of processed jobs). We denote by  $m_j(t)$  the number of servers allocated to type- $j$  jobs at time  $t$ . In our study, we focus on server power consumption, while the power consumption of other parts such as power supply and cooling systems can be captured by the power usage effectiveness (PUE) factor which, multiplied by the server power consumption, gives the total data center power consumption [67]. Mathematically, we denote the total power consumption<sup>1</sup> during time  $t$  by  $p(\lambda(t), \mathbf{m}(t))$ , which can be expressed as

$$p(\lambda(t), \mathbf{m}(t)) = \gamma \cdot \sum_{j=1}^J m_j(t) \cdot \left[ e_0 + e_c \frac{\lambda_j(t)}{m_j(t) \mu_j} \right], \quad (3.1)$$

where  $\gamma > 1$  is the PUE,  $e_0$  is the static server power regardless of the workloads (as long as a server is turned on) and  $e_c$  is the computing power incurred only when a server is processing workloads.

We consider that the data center participates in smart grid using (hourly) time-varying demand-responsive electricity prices [107, 117]. Assuming that the amount of available on-site renewable energy is  $r(t)$  as specified in the next subsection, we can express the incurred electricity cost during time  $t$  as

$$e(\lambda(t), \mathbf{m}(t)) = u(t) \cdot [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+, \quad (3.2)$$

---

<sup>1</sup>This is equivalent to energy consumption, since the length of each time slot is the same.

where  $[\cdot]^+ = \max\{\cdot, 0\}$  indicating that no electricity will be drawn from the power grid if on-site renewable energy is already sufficient, and  $u(t)$  is the demand-responsive electricity price specified as follows.

### 3.2.3 Demand-responsive electricity price

In smart grid, electricity prices are updated periodically based on the total energy demand, including the energy consumption of both the data center and other energy consumers (e.g., households) in the local electricity market. As in [107, 117], we assume that the data center operator knows the total energy demand of non-data-center consumers, denoted by  $b(t)$ , which may be obtained by accessing the public data provided by the utility company or learnt based on history traces. Then, the data center can estimate the demand-responsive electricity price (by considering the impact of its server capacity provisioning decisions) and communicate its energy consumption signal to the utility company, which will then set the electricity price accordingly.<sup>2</sup> While small energy consumers (e.g., households) may also re-adjust its energy demand based on the electricity price, we assume for simplicity that the total energy demand of non-data-center consumers is known at the beginning of each time slot and does not change within each time slot [107, 117]. Note that considering strategic energy consumption decisions made by non-data-center consumers requires a game-theoretic approach [84] and may be a potential area to explore in the future. Next, we provide a concrete example to explain how to model the impact of data center energy consumption on the electricity price.

We first plot in Figs. 3.1(a) and 3.1(b) the energy demands and hourly electricity prices during the first 15 days of 2012 for one of three electricity zones serving Mountain

---

<sup>2</sup>With a large energy consumer such as data center whose energy demand has a significant impact on electricity generation and distribution, it is likely that the electricity price is determined by taking into account the (dynamic) decisions by the large energy consumer, as corroborated by [107, 117]. Thus, in this part of the dissertation, we take the liberty of using demand-responsive electricity prices for the data center.

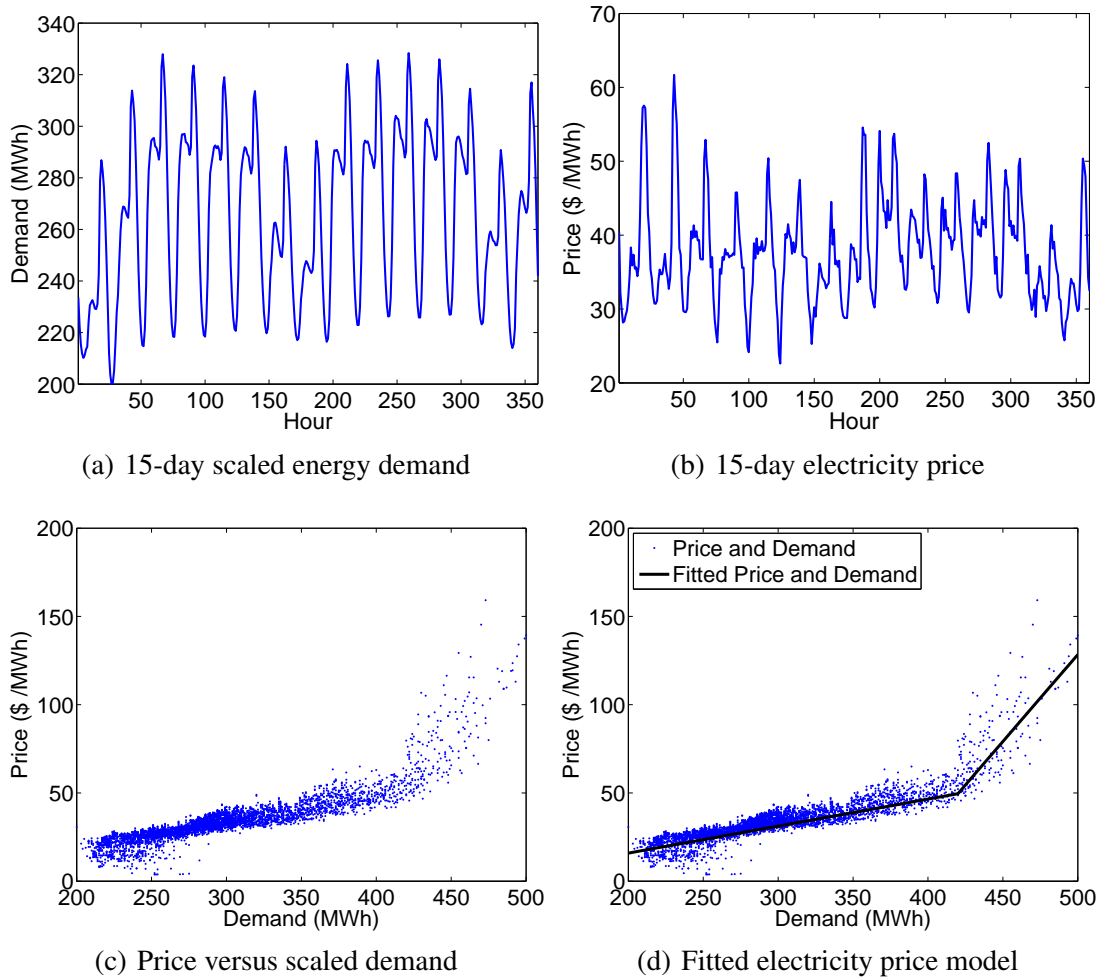


Figure 3.1: Electricity trace data

View, California. Due to the lack of access to exact data, we obtain the hourly electricity demand by scaling the total demand of California in proportion to the population ratio of Mountain View (divided by three, due to three electricity zones serving the city) to California [3]. As intuitively expected, the general trend is that a higher demand leads to a higher electricity price, as further corroborated by Fig. 3.1(c) that contains the 6-month data from January to June, 2012. Then, by applying mean square error data fitting to the 6-month data in Fig. 3.1(c), we can approximate electricity price using a piecewise linear

function as

$$u(t) = \begin{cases} 0.15[a(t) + b(t)] - 15.6, & \text{if } 200 \leq a(t) + b(t) \leq 420, \\ 0.98[a(t) + b(t)] - 364.2, & \text{if } a(t) + b(t) > 420, \end{cases} \quad (3.3)$$

where  $a(t)$  is the total energy demand of non-data-center consumers and  $b(t) = [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+$  is the electricity usage by the data center. The fitted function and original data are shown in Fig. 3.1(d). Note that, despite having different parameters, similar piecewise linear electricity price functions have also been reported in [107, 117]. From Fig. 3.1(d), we notice that a cloud-scale data center with a peak power of 50MW and an average utilization of 40% can increase the electricity price by more than 10%, highlighting the importance of incorporating demand-responsive electricity price in data center operation. While the fitted electricity function may not fully characterize the demand-price relation, we will show using extensive simulations that even with inaccurate knowledge of non-data-center energy demand  $b(t)$  (with 10% errors), the data center can leverage the demand-responsive electricity price to reduce its operational cost by more than 2%, translating into a significant cost saving in practice. Finally, we note that our demand-responsive price model can also be applied to enforce peak demand shaving, which is usually employed to reduce peak power usage charge [69]. Specifically, if the price function is appropriately modified such that a very high price is charged when the data center's electricity usage exceeds a certain threshold (as similarly considered in [69]), then the peak power usage fee can be effectively mitigated.

### 3.2.4 Renewable energy

We consider the following three representative types of renewable energy sources that have been increasingly adopted by large data centers [27, 41]. Note that we ignore the DC-AC energy conversion loss and hence, the values in this chapter denote the net available renewable energy for powering the data center.

**On-site renewable energy:** Renewable energy generators such as solar panels and wind turbines can be easily installed on-site and directly provide green energy to power data centers. Typically, on-site renewable energy supplies are highly dependent on weather conditions, exhibiting an intermittent nature. We denote the available on-site renewable energy supply during time  $t$  by  $r(t) \in [0, r_{\max}]$ , which may follow an arbitrary trajectory throughout the budgeting period. While energy storage devices such as batteries can be utilized as a complementary solution for preventing power outages [42] and storing the unused renewable energy to smooth the electricity usage for cost saving [104, 106], we do not consider it in our model because: (1) these devices are quite expensive to install at a large scale [82]; and (2) our main focus is on making resource management decisions rather than charging/discharging batteries.

**Off-site renewable energy:** As the best locations for renewable energy production may not be good for building a data center, large data center operators are now using off-site renewable energy to achieve carbon neutrality [27, 41]. One important and widely-used type of off-site renewable energy is through PPAs. For example, Google has invested in and signed PPAs with several renewable energy plants such that the generated renewable energy will be directly fed into the local electricity grid and then used to indirectly offset the brown energy (i.e., electricity) usage of Google’s data centers [41]. We denote the supply of the off-site renewable energy generated via PPAs for time  $t$  as  $f(t) \in [0, f_{\max}]$ .

**REC:** As a complementary source, RECs may be used to offset data centers’ brown energy usage. Two popular approaches are available to obtain RECs: (1) purchase a large amount of RECs at the beginning of a budgeting period; and (2) dynamically purchase RECs for each time slot from the REC market. We focus on the first approach and denote the (fixed) amount of RECs available for the data center throughout the budgeting period by  $Z$ . While some companies may purchase RECs at the end of a budgeting period to

offset the remaining brown energy usage, this approach is orthogonal to our study: remaining brown energy, if any, may still be offset by purchasing additional RECs at the end of a budgeting period.

### 3.3 Algorithm Design and Analysis

In this section, we present the offline problem formulation and develop an online algorithm, CNDC, which explicitly incorporates demand-responsive electricity prices and is provably efficient in terms of cost minimization compared to the optimal algorithm with lookahead information. CNDC builds upon yet extends the recently developed Lyapunov optimization technique [87].

#### 3.3.1 Problem formulation

This subsection presents an offline problem formulation for capacity provisioning under carbon neutrality.

We first define operational cost as the objective function. Specifically, both electricity cost and delay cost are important for data centers, as the former takes up a dominant fraction of the operational cost while the latter affects the user experiences and revenues [66]. Our study incorporates both costs by considering a parameterized cost function as follows<sup>3</sup>

$$g(\lambda(t), \mathbf{m}(t)) = e(\lambda(t), \mathbf{m}(t)) + \beta \cdot d(\lambda(t), \mathbf{m}(t)), \quad (3.4)$$

where  $\beta \geq 0$  is the weighting parameter adjusting the importance of delay cost relative to the electricity cost [68]. The objective is to minimize the long-term average cost expressed

---

<sup>3</sup>Although off-site renewable energy supplies are not free, the payment is often subject to PPAs and not affected by data center resource management decisions [27, 41]. Thus, we do not consider it as the data center's *operational* cost.



as

$$\bar{g} = \frac{1}{K} \sum_{t=0}^{K-1} g(\lambda(t), \mathbf{m}(t)), \quad (3.5)$$

where  $K$  is the total number of time slots over the entire budgeting period. Next, we formulate the offline capacity provisioning problem below.

$$\mathbf{P11} : \quad \min_{\mathcal{A}} \bar{g} = \frac{1}{K} \sum_{t=0}^{K-1} g(\lambda(t), \mathbf{m}(t)) \quad (3.6)$$

$$s.t., \quad \lambda_j(t) \leq \theta \cdot \mu_j \cdot m_j(t), \quad \forall j, t, \quad (3.7)$$

$$\sum_{j=1}^J m_j(t) \leq M, \quad \forall t, \quad (3.8)$$

$$\frac{1}{K} \sum_{t=0}^{K-1} b(t) \leq \frac{\alpha}{K} \left[ \sum_{t=0}^{K-1} f(t) + Z \right], \quad (3.9)$$

where  $b(t) = [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+$  is the electricity usage and  $\mathcal{A}$  represents a sequence of capacity provisioning decisions, i.e.,  $\mathbf{m}(t)$ , for  $t = 0, 1, \dots, K-1$ , which we need to optimize. The constraints (3.7) and (3.8) are imposed to avoid server overloading and over-provisioning, respectively, where  $\theta \in (0, 1)$  is the maximum utilization constraint for each server (i.e.,  $\frac{\lambda_j(t)}{\mu_j \cdot m_j(t)} \leq \theta$ ). The constraint (3.9) specifies the long-term carbon neutrality: following the current industry practice [27, 41, 68], we say that a data center achieves carbon neutrality as long as its electricity usage is completely *offset* by the off-site renewable energy plus RECs. While we express carbon neutrality as a hard constraint, it is actually a desired target/goal: even though carbon neutrality cannot be possibly satisfied (e.g., due to persistently high workloads), the data center still continues processing workloads rather than dropping them. The parameter  $\alpha > 0$  in (3.9) indicates the desired offsetting of electricity usage relative to the total off-site renewable energy plus RECs: the less  $\alpha$ , more aggressive the data center is in achieving carbon neutrality.

It is natural that optimally solving **P11** requires complete offline information (i.e., workload arrivals, renewable energy supplies, and electricity prices) that is very difficult,

even impossible, to accurately predict in advance, especially in view of the unpredictability of weather conditions that heavily affect the renewable energy availability [67]. In what follows, we propose an online algorithm in which capacity provisioning decisions are made based on the hour-ahead information only.

### 3.3.2 CNDC

We note first that the long-term carbon neutrality constraint couples the capacity provisioning decisions across different time slots. Thus, eliminating the carbon neutrality constraint will lead to an online solution. Towards this end, as the foundation of CNDC, we construct a (virtual) carbon deficit queue that *replaces* the long-term constraint (3.9) and decouples the decisions for different time slots. Specifically, assuming  $q(0) = 0$ , we construct a carbon deficit queue whose dynamics evolves as follows

$$q(t+1) = \{q(t) + [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ - \alpha \cdot f(t) - z\}^+, \quad (3.10)$$

where  $q(t)$  is the queue length indicating how far the current electricity usage deviates from the carbon neutrality constraint, and  $z = \frac{\alpha}{K} \cdot Z$  is the average RECs per time slot scaled by  $\alpha$ . Intuitively, a large queue length implies that the data center has drawn more electricity than the total off-site renewable energy plus RECs provided thus far, and it needs to reduce the electricity usage for carbon neutrality. Leveraging this intuition, we develop our online algorithm, CNDC, as presented in Algorithm 1.

In Algorithm 1, we use  $V_0, V_1, \dots, V_{R-1}$  to denote a sequence of positive control parameters (also referred to as cost-carbon parameters) to dynamically adjust the tradeoff between cost minimization and electricity usage over the  $R$  frames, each having  $T$  time slots. Lines 2-4 reset the carbon deficit queue at the beginning of each frame  $r$ , such that the cost-carbon parameter  $V$  can be adjusted and the carbon deficit in a new time frame will not be affected by its value resulting from the previous time frame. Line 5 defines an

---

**Algorithm 1: CNDC**

---

- Input:** Input  $\lambda(t), r(t)$  and  $a(t)$  at the beginning of each time  $t = 0, 1, \dots, K - 1$ .
- 1 **if**  $t = rT, \forall r = 0, 1, \dots, R - 1$  **then**
  - 2      $q(t) \leftarrow 0$  and  $V \leftarrow V_r$
  - 3 Choose  $\mathbf{m}(t)$  subject to (3.7)(3.8) to minimize

$$V \cdot g(\lambda(t), \mathbf{m}(t)) + q(t) \cdot [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ \quad (3.11)$$

- 4 Update  $q(t)$  according to (3.10).
- 

optimization problem to decide  $\mathbf{m}(t)$  based on online information: minimizing the original cost scaled by  $V$  plus  $q(t) \cdot [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+$ . By considering the perturbing term  $q(t) \cdot [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+$ , the data center operator places a higher weight on the electricity usage while making capacity provisioning decisions if the queue length  $q(t)$  is larger: as a consequence, when  $q(t)$  increases, minimizing the electricity usage is more critical for the data center operator due to the carbon neutrality constraint. In essence, the carbon deficit queue maintained without foreseeing the future guides the data center decisions towards carbon neutrality.

Now, we explain the impact of the cost-carbon parameter  $V$  on online decisions, as will be formalized in the next subsection.

- When  $V$  becomes larger, the data center tends to minimize the cost, while the carbon neutrality constraint plays a less important role and the carbon queue length makes a significant impact on online decisions only when the electricity usage has deviated too much from carbon neutrality (i.e.,  $q(t)/V$  cannot be neglected).

- When  $V$  becomes smaller, the data center tends to follow carbon neutrality more closely while caring less about the cost, as the carbon deficit queue length plays a more significant role when the data center makes decisions (i.e.,  $q(t)/V$  will easily become large).

### 3.3.3 Performance analysis

This subsection presents the performance analysis of CNDC: it achieves a long-term average cost arbitrarily close to that achieved by the optimal algorithm with lookahead information while the deviation from carbon neutrality is bounded.

We first introduce an offline algorithm with lookahead information as a benchmark. Specifically, we divide the entire budgeting period into  $R$  frames, each having  $T \geq 1$  time slots, such that  $K = RT$ . Then, at the beginning of the  $r$ -th frame, for  $r = 0, 1, \dots, R-1$ , offline decisions are chosen to solve the following problem:

$$\mathbf{P12} : \min_{\mathbf{m}(t)} \frac{1}{T} \sum_{t=rT}^{(r+1)T-1} g(\lambda(t), \mathbf{m}(t)) \quad (3.12)$$

$$s.t., \quad \text{constraints (3.7), (3.8),} \quad (3.13)$$

$$\sum_{t=rT}^{(r+1)T-1} [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ \leq \alpha \cdot f_r, \quad (3.14)$$

where  $f_r = \sum_{t=rT}^{(r+1)T-1} f(t) + \frac{Z}{R}$  is the total amount of available off-site renewable energy supplies during the  $r$ -th frame plus the total RECs evenly distributed over the  $R$  frames. In essence, **P12** encapsulates a *family* of offline algorithms parameterized by the lookahead information window size  $T$ . Next, to ensure there exists at least one feasible solution to **P12**, we make the two assumptions that are very mild in practice.

*Boundedness assumption:* The workload arrival rate  $\lambda(t)$ , electricity price  $u(t)$ , as well as renewable energy supplies  $r(t)$  and  $f(t)$  are finite, for  $t = 0, 1, \dots, K-1$ .

*Feasibility assumption:* For the  $r$ -th frame, where  $r = 0, 1, \dots, R-1$ , there exists at least one sequence of capacity provisioning decisions that satisfy the constraints of **P12**.

The boundedness assumption ensures that the cost function is finite, while the feasibility assumption guarantees that the oracle can make a sequence of feasible decisions to solve **P12**. We denote the minimum average cost for the  $r$ -th frame by  $G_r^*$ , for  $r = 0, 1, \dots, R$ , considering all the decisions that satisfy the constraints (3.13)(3.14) and that have perfect information over the frame.

Next, building upon yet extending the recently-developed Lyapunov optimization technique [87], we formalize the performance analysis of CNDC in Theorem 1, whose proof outline is available in the appendix.

**Theorem 1.** *Suppose that boundedness and feasibility assumptions are satisfied. Then, for any  $T \in \mathbb{Z}^+$  and  $R \in \mathbb{Z}^+$  such that  $K = RT$ , the following statements hold.*

*a. The carbon neutrality constraint is approximately satisfied with a bounded deviation:*

$$\begin{aligned} & \frac{1}{K} \sum_{t=0}^{K-1} [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ \\ & \leq \frac{\alpha}{K} \cdot \left[ \sum_{t=0}^{K-1} f(t) + Z \right] + \frac{\sum_{r=0}^{R-1} \sqrt{C(T) + V_r (G_r^* - g_{\min})}}{R\sqrt{T}}, \end{aligned} \quad (3.15)$$

where  $C(T) = B + D(T - 1)$  with  $B$  and  $D$  being finite constants,  $G_r^*$  is the minimum average cost achieved over the  $r$ -th frame by the optimal offline algorithm with  $T$ -slot lookahead information, for  $r = 0, 1, \dots, R - 1$ , and  $g_{\min}$  is the minimum hourly cost that can be achieved by any feasible decisions throughout the budgeting period.

*b. The average cost  $\bar{g}^*$  achieved by CNDC satisfies:*

$$\bar{g}^* \leq \frac{1}{R} \sum_{r=0}^{R-1} G_r^* + \frac{C(T)}{R} \cdot \sum_{r=0}^{R-1} \frac{1}{V_r}. \quad (3.16)$$

We prove Theorem 1 following three key steps:

1. We present Lemma 1 that relates the carbon queue length to *approximate* constraint satisfaction.
2. We define a quadratic Lyapunov function for the carbon deficit queue length and derive upper bounds on the one-slot as well as  $T$ -slot Lyapunov drift plus cost.
3. We minimize the derived upper bounds using CNDC and then compare with the optimal offline algorithm with  $T$ -step lookahead information to complete the proof.

**Lemma 1.** For any  $0 \leq r \leq R-1$ , any carbon deficit queue length  $q(rT)$ , and any feasible decision satisfying (3.7)(3.8), we have

$$\begin{aligned} & \frac{1}{T} \sum_{t=rT}^{(r+1)T-1} [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ \\ & \leq \frac{\alpha}{T} \left[ \sum_{t=rT}^{(r+1)T-1} f(t) + \frac{Z}{R} \right] + \frac{q(rT+T) - q(rT)}{T}. \end{aligned} \quad (3.17)$$

*Proof.* Following the carbon deficit queue dynamics specified by (3.10), we have for any  $t \in [rT, rT+T-1]$  and any  $r = 0, 1, \dots, R-1$ :

$$q(t+1) - q(t) \geq [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ - \alpha \cdot f(t) - z, \quad (3.18)$$

where  $z = \frac{\alpha}{J} \cdot Z$  is the average RECs per time slot scaled by  $\alpha$ . By summing over  $t = rT, rT+1, \dots, rT+T-1$ , we obtain

$$\begin{aligned} & q(rT+T) - q(rT) \\ & \geq \sum_{t=rT}^{rT+T-1} \{ [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+ - \alpha \cdot f(t) - z \}. \end{aligned} \quad (3.19)$$

Then, (3.17) is proved by rearranging the terms and dividing both sides of (3.19) by  $T$ . ■

Lemma 1 shows that the carbon neutrality constraint can be *approximately* satisfied over the  $r$ -th frame if the carbon queue length difference  $q(rT+T) - q(rT)$  is sufficiently small. Next, we define for notational convenience

$$y(t) = [p(\lambda(t), \mathbf{m}(t)) - r(t)]^+, \quad (3.20)$$

$$z(t) = \alpha \cdot f(t) + z, \quad (3.21)$$

$$g(t) = g(\lambda(t), \mathbf{m}(t)). \quad (3.22)$$

Thus, (3.10) can be rewritten as  $q(t+1) = [q(t) + y(t) - z(t)]^+$ . As an alternative scalar measure of all the queue lengths, we also define the quadratic Lyapunov function

$$L(q(t)) \triangleq \frac{1}{2} q^2(t), \quad (3.23)$$

where the scaling factor  $\frac{1}{2}$  is added for the convenience of analysis. Let  $\Delta_T(t)$  be the  $T$ -slot Lyapunov drift yielded by some control decisions over the interval  $t, t+1, \dots, t+T-1$ :

$$\Delta_T(t) \triangleq L(q(t+T)) - L(q(t)). \quad (3.24)$$

Similarly, the 1-slot drift is

$$\Delta_1(t) \triangleq L(q(t+1)) - L(q(t)). \quad (3.25)$$

Based on  $q(t+1) = [q(t) + y(t) - z(t)]^+$ , it can be shown that  $L(q(t+T)) = \frac{1}{2}q^2(t+1) \leq \frac{1}{2}[q(t) + y(t) - z(t)]^2$ . Then, it can be shown that the 1-slot drift satisfies

$$\Delta_1(t) \leq B + q(t) \cdot [y(t) - z(t)], \quad (3.26)$$

where  $B$  is a constant satisfying, for all  $t = 0, 1, \dots, J-1$ ,

$$B \geq \frac{1}{2}[y(t) - z(t)]^2, \quad (3.27)$$

where is finite due to the boundedness assumption.

Based on the inequality in (3.26), we can easily show

$$\Delta_1(t) + V \cdot g(t) \leq B + V \cdot g(t) + q(t) \cdot [y(t) - z(t)]. \quad (3.28)$$

The online algorithm described in line 5 of Algorithm 1 actually minimizes the upper bound on the 1-slot Lyapunov drift plus a weighted cost shown on the right hand side of (3.28). Following (3.28), we can show that, for  $r = 0, 1, \dots, R-1$ , the  $T$ -slot drift plus

weighted cost satisfies

$$\begin{aligned}
& \Delta_T(rT) + V_r \sum_{t=rT}^{rT+T-1} g(t) \\
& \leq BT + V_r \sum_{t=rT}^{rT+T-1} g(t) + \sum_{t=rT}^{rT+T-1} q(t) \cdot [y(t) - z(t)] \\
& \leq BT + V_r \sum_{t=rT}^{rT+T-1} g(t) + \sum_{t=rT}^{rT+T-1} (t - rT) q^{diff} \cdot |y(t) - z(t)| \\
& \quad + q(rT) \sum_{t=rT}^{rT+T-1} [y(t) - z(t)] \\
& \leq BT + V_r \sum_{t=rT}^{rT+T-1} g(t) + DT(T - 1) + q(rT) \sum_{t=rT}^{rT+T-1} [y(t) - z(t)],
\end{aligned} \tag{3.29}$$

where  $q^{diff} = \max_{t=0,1,\dots,J-1} \{y(t), z(t)\}$  and  $D$  is a finite constant satisfying  $D \geq \frac{1}{2}q^{diff} \cdot \max\{y(t), r(t)\}$ . Note that CNDC explicitly minimizes the term  $BT + V_r \sum_{t=rT}^{rT+T-1} g(t) + \sum_{t=rT}^{rT+T-1} q(t) \cdot [y(t) - z(t)]$ . Thus, by applying CNDC on the left-hand side and considering the optimal  $T$ -step lookahead policy on the right-hand side of (3.29), we obtain the following inequality

$$\begin{aligned}
& \Delta_T(rT) + V_r \sum_{t=rT}^{rT+T-1} g^*(t) \\
& \leq BT + V_r T G_r^* + DT(T - 1) + q(rT) \sum_{t=rT}^{rT+T-1} [y(t) - z(t)] \\
& \leq BT + V_r T G_r^* + DT(T - 1),
\end{aligned} \tag{3.30}$$

where  $g^*(t)$  is the cost achieved by CNDC at time  $t$  and the second inequality follows from the constraints in (3.14) satisfied by the optimal  $T$ -slot lookahead policy. Note that  $q(rT)$  is reset to zero, for  $r = 0, 1, \dots, R - 1$ , as enforced by Algorithm 1, whereas  $\Delta_T(rT) = q^2(rT + T) - q^2(rT) = q^2(rT + T)$  in (3.30) is the  $T$ -step Lyapunov drift calculated *after* the  $r$ -th reset but *before* the  $(r + 1)$ -th reset of the carbon deficit queue.



Thus, before the  $(r + 1)$ -th reset of the carbon deficit queue, we obtain from (3.30)

$$\begin{aligned}
q(rT + T) &\leq \sqrt{BT + DT(T - 1) + V_r T G_r^* - V_r \sum_{t=rT}^{rT+T-1} g^*(t)} \\
&\leq \sqrt{BT + DT(T - 1) + V_r T (G_r^* - g_{\min})} \\
&= \sqrt{T} \cdot \sqrt{B + D(T - 1) + V_r (G_r^* - g_{\min})},
\end{aligned} \tag{3.31}$$

where  $g_{\min}$  is the minimum cost that can be achieved by any feasible decisions throughout the budgeting period. Then, by Lemma 1, we derive

$$\frac{1}{T} \sum_{t=rT}^{(r+1)T-1} y(t) \leq \frac{1}{T} \sum_{t=rT}^{(r+1)T-1} z(t) + \frac{\sqrt{C(T) + V_r (G_r^* - g_{\min})}}{\sqrt{T}}, \tag{3.32}$$

where we define

$$C(T) = B + D(T - 1). \tag{3.33}$$

Therefore, by summing (3.32) over  $r = 0, 1, \dots, R - 1$  and dividing both sides by  $R$ , we obtain

$$\frac{1}{J} \sum_{t=0}^{J-1} y(t) \leq \frac{1}{J} \sum_{t=0}^{J-1} z(t) + \frac{\sum_{r=0}^{R-1} \sqrt{C(T) + V_r (G_r^* - g_{\min})}}{R\sqrt{T}}, \tag{3.34}$$

which proves part (a) of Theorem 1.

Next, by dividing both sides of (3.30) by  $V_r$  and considering  $q(rT) = 0$  as enforced by Algorithm 1, it follows that

$$\begin{aligned}
\sum_{t=rT}^{rT+T-1} g^*(t) &\leq \frac{BT}{V_r} + T G_r^* + \frac{DT(T - 1)}{V_r} - \frac{\Delta_T(rT)}{V_r} \\
&\leq \frac{BT + DT(T - 1)}{V_r} + T G_r^*.
\end{aligned} \tag{3.35}$$

Finally, by summing (3.35) over  $r = 0, 1, \dots, R - 1$  and dividing both sides by  $RT$ , we have

$$\bar{g}^* \leq \frac{1}{R} \sum_{r=0}^{R-1} G_r^* + \frac{B + D(T - 1)}{R} \cdot \sum_{r=0}^{R-1} \frac{1}{V_r}, \tag{3.36}$$

which shows that the online algorithm can achieve a cost within  $O(1/V)$  to the minimum cost achieved by the optimal offline algorithm with  $T$ -step lookahead information. This proves part (b) of Theorem 1. ■

Theorem 1 shows that, given a fixed value of  $T$  and  $R$ , CNDC is  $O(1/V)$ -optimal with respect to the average cost against the optimal  $T$ -step lookahead policy, i.e., CNDC incurs no more than  $O(1/V)$  additive cost than the minimum value, while the carbon neutrality constraint is guaranteed to be *approximately* satisfied with a bounded “fudge factor” of  $\frac{\sum_{r=0}^{R-1} \sqrt{C(T) + V_r(G_r^* - g_{\min})}}{R\sqrt{T}}$ . With a larger  $V$ , the cost is closer to the minimum but the potential deviation of electricity usage from carbon neutrality can be larger, and vice versa.

Next, we discuss how the performance result is affected by the value of  $T$ . Considering  $K = RT$  in (3.15), CNDC incurs an average electricity usage closer to carbon neutrality as  $T$  increases, which can also be formally shown by rewriting the fudge factor in (3.15) but the details are omitted due to space limitations. This is because the carbon deficit queue length is reset to zero less frequently (i.e., lines 2–4 in Algorithm 1) and hence, the carbon deficit queue is more likely to be non-empty, thereby giving the data center a higher pressure on reducing its electricity usage. On the other hand, when  $T$  becomes smaller, the carbon deficit will be cleared more frequently, and accordingly CNDC uses electricity more *aggressively* with less restriction imposed by the carbon deficit queue. Meanwhile, as  $T$  increases (i.e., the oracle can further look into the future and thus can make a better decision), the bounded gap between the average cost achieved by CNDC and that achieved by the optimal  $T$ -step lookahead policy also increases.

While Theorem 1 strictly holds under boundedness and feasibility assumptions that are very mild in practice, CNDC applies to an arbitrarily changing environment even though the two assumptions are not satisfied. To derive tighter analytical bounds, additional assumptions on the environment dynamics (e.g., i.i.d./Markovian workload arrival

rate, renewable energy supply and electricity price [46, 87]) are required which, however, may not hold in practice.

### 3.4 Simulation Study

This section presents trace-based simulation studies of a large data center to validate our analysis and evaluate the performance of CNDC. We first present our data sets and then show the following sets of simulations:

- The impact of  $V$ : We show how cost minimization and satisfaction of carbon neutrality varies with different values of  $V$  as well as the impact of dynamically changing  $V$  over the course of operation.
- Comparison with prediction-based method: We compare CNDC with the state-of-the-art prediction-based method and show that CNDC reduces the average cost by more than 20% while resulting in a less carbon footprint.
- Comparison with non-demand-responsive electricity price: We compare CNDC with an algorithm that does not incorporate demand-responsive electricity prices and show that CNDC achieves more than 2.5% cost saving.
- Sensitivity study: We show that CNDC still yields a satisfactory performance in terms of the operational cost while satisfying carbon neutrality: (1) in the presence of workload prediction errors or inaccurate knowledge of non-data-center energy demand; and (2) when using measured real server power consumption, workload delay, and toggling energy consumption.

#### 3.4.1 Data sets

We consider a data center with a peak power of 50MW and an average PUE of 2.0.<sup>4</sup> The data center consists of 100,000 servers in total, each with a maximum power of 250W

---

<sup>4</sup>State-of-the-art techniques have reduced this value of around 1.12 [41].

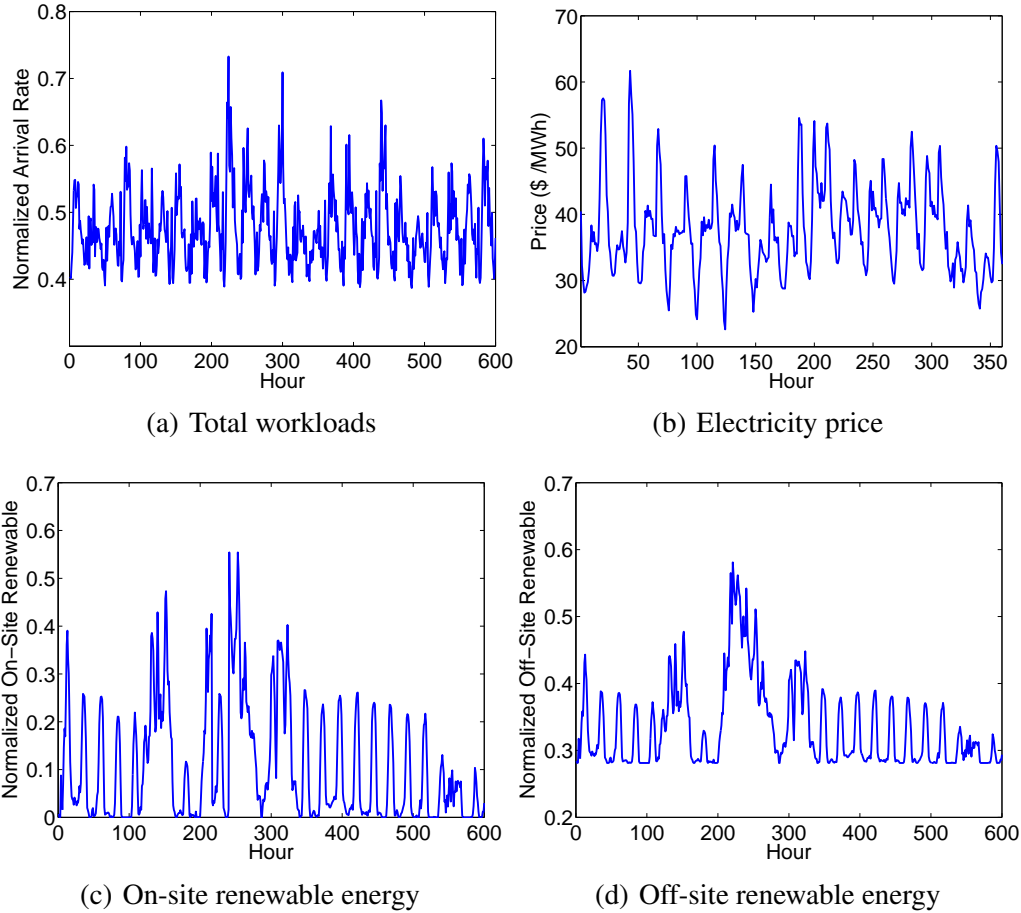


Figure 3.2: Trace data for January 1–25, 2012.

and idle power of 150W. As in the existing work [68,93], we only model the server power consumption for delay-sensitive workloads as our main focus without considering delay-tolerant batch jobs. The parameter converting the delay cost into monetary cost is set as  $\beta = 0.03$ , while CNDC is applicable for any value of  $\beta \geq 0$ . The budgeting period in our study is 6 months.

- **Workloads:** We use three different types of workloads with equal weights in the delay cost (i.e.,  $w_1 = w_2 = w_3 = 1$ ). We profile the server usage log of Florida International University (FIU, a large public university in the U.S. with over 50,000 students) from January 1 to June 30, 2012. We also plot workload traces for Microsoft Research and

Hotmail shown in [66] and repeat the traces for 6 months by adding random noises of up to  $\pm 40\%$ . The normalized service rates of each server for these three workloads are 0.95, 1.00 and 1.05, respectively. Due to the lack of access to workloads of large commercial data centers, we scale these workloads and show in Fig. 3.2(a) the trace of January 1–25, 2012, normalized with respect to the total service rates provided by the data center. Note that our trace exhibits a similar pattern with those of large commercial data centers (e.g., Facebook trace shown in [94]).

- Renewable energy: We obtain from [3] the hourly renewable energies (generated through solar panels and wind turbines) for the city of Mountain View as well as the state of California during the first six months of 2012. We scale the data proportionally such that on average on-site and off-site renewable energies satisfy 15% and 35% of the peak power consumption, respectively. Figs. 3.2(c) and 3.2(d) show the available on-site and off-site renewable energies during January 1–25, 2012, respectively, normalized with respect to the peak power of the data center (50MW). We set the available RECs as approximately 16% of total energy demand.

- Electricity price: With demand-responsive electricity prices, we use the fitted piecewise linear function shown in (3.3), while the non-data-center energy demand is obtained by subtracting the scaled demand (as illustrated in Fig. 3.1(a)) by 50MWh. As a comparison, if demand-responsive electricity price is not incorporated, we directly use the hourly electricity price for a trading node in Mountain View, California, obtained from [3].

Since the access to commercial data centers is unavailable, we obtain the trace data from various sources, but it captures the variation of workload, renewable energy supply and electricity prices over the budgeting period. Thus, it serves the purpose of evaluating the performance and benefits of CNDC. Next, we present simulation results using the above trace data.

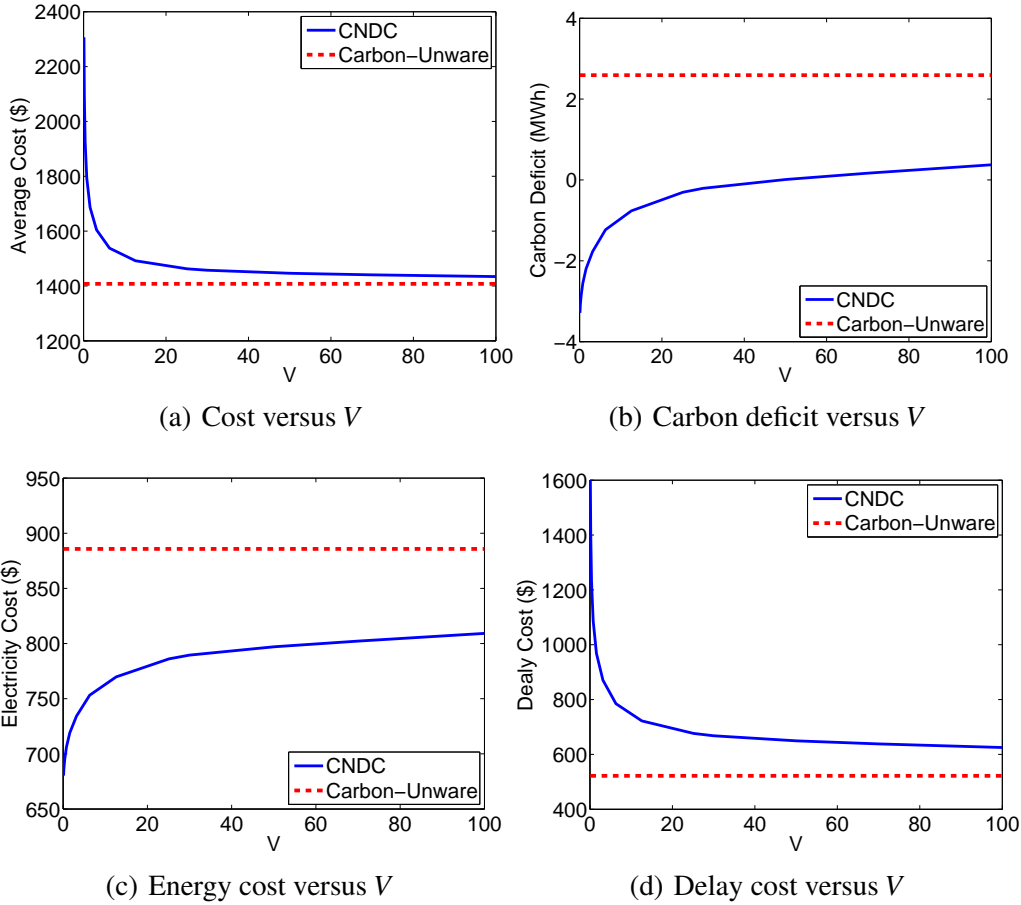


Figure 3.3: Impact of  $V$ .

### 3.4.2 Impact of $V$

We now show how the carbon-cost  $V$  affects the performance of CNDC.

**Constant  $V$ .** We first consider a constant  $V$  throughout the budgeting period. Fig. 3.3(a) and Fig. 3.3(b) show the impact of  $V$  on the average hourly cost (i.e.,  $\bar{g}$ ) and average hourly carbon deficit (i.e., average hourly electricity usage minus the available carbon budget consisting of both off-site renewable energy and RECs), respectively. Note that carbon deficit may be either positive or negative, depending on the amount of off-site renewable energies plus RECs: negative deficit means off-site renewable energies plus RECs exceed the electricity usage. The result conforms with our analysis that with a greater  $V$ , CNDC

is less concerned with the carbon deficit while caring more about the cost. The reason is that, with a large value of  $V$ , the weight of carbon deficit in the optimization objective (5.9) is relatively smaller, thereby equivalently making the carbon neutrality constraint less stringent. This can also be seen from Fig. 3.3(c) and Fig. 3.3(d) that show the average hourly electricity cost and delay cost, respectively: when  $V$  increases, the data center turns on more servers, leading to a better delay performance while resulting more carbon footprint as well as electricity cost. In the extreme case in which  $V$  goes to infinity, CNDC reduces to a *carbon-unaware* algorithm that minimizes the cost without considering carbon neutrality. Clearly, carbon-unaware algorithm achieves a cost that is a lower bound on the cost that can be possibly achieved by any algorithm satisfying the carbon neutrality constraint. Nonetheless, the carbon-unaware algorithm is likely to violate the carbon neutrality constraint due to the insufficient supply of off-site renewable energies or RECs.<sup>5</sup> It can be seen from Fig. 3.3(a) and Fig. 3.3(b) that the cost achieved by CNDC is fairly close to the lower bound on the cost achieved by the carbon-unaware algorithm when  $V$  exceeds 30, whereas CNDC satisfies the carbon neutrality constraint for  $V \leq 30$ . This indicates that, with  $V \approx 30$ , CNDC achieves a close-to-minimum cost while still satisfying the carbon neutrality constraint.

**Varying  $V$ .** We show in Fig. 3.4 the impact of dynamically changing  $V$  over the course of operation. Specifically, we change  $V$  every 1.5 months and present the moving average hourly cost and carbon deficit (averaged over the past 360 hours) in Fig. 3.4(a) and Fig. 3.4(b), respectively. The fluctuation of moving average values is mainly due to the large variation of workloads. We observe from Fig. 3.4 that, by choosing a small  $V$  initially, the average cost is quite big whereas it can be significantly reduced later by increasing the value of  $V$  (at the expense of increasing the carbon deficit). This indicates

---

<sup>5</sup>If there are always sufficient off-site renewable energies or RECs, carbon neutrality constraint in (3.9) can be safely removed. Thus, the data center can use any amount of electricity that will be offset by off-site renewable energies or RECs, which makes the problem trivial.

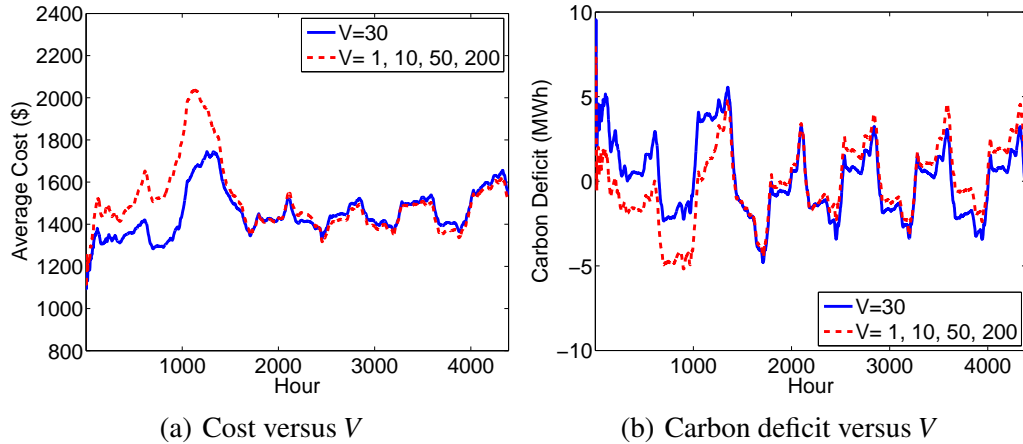


Figure 3.4: Impact of time-varying  $V$ .

the flexibility of dynamically tuning  $V$  to adjust the tradeoff between cost minimization and the potential violation of carbon neutrality.

In Figs. 3.3 and 3.4, we do not show the optimal offline algorithm with  $T$ -step lookahead information, because it cannot possibly achieve a cost less than the optimal carbon-unaware algorithm, compared to which CNDC already achieves a close-to-minimum cost.

### 3.4.3 Comparison with prediction-based method

We now compare the performance of CNDC with the best known existing solution studied in [64, 94, 117]. However, since none of these research considered both the nonlinear delay cost and intermittent off-site renewable energy supplies, we incorporate these factors to the existing prediction-based method [64, 94, 117] and consider the following heuristic variation.

- Perfect hourly prediction heuristic (**PerfectHP**): The data center operator predicts the next-day workload perfectly and allocates the daily carbon budget in proportion to the hourly workloads.

In PerfectHP, the daily carbon budget are obtained by dividing the total RECs evenly across each day and then adding the total next-day off-site renewable energy supplies



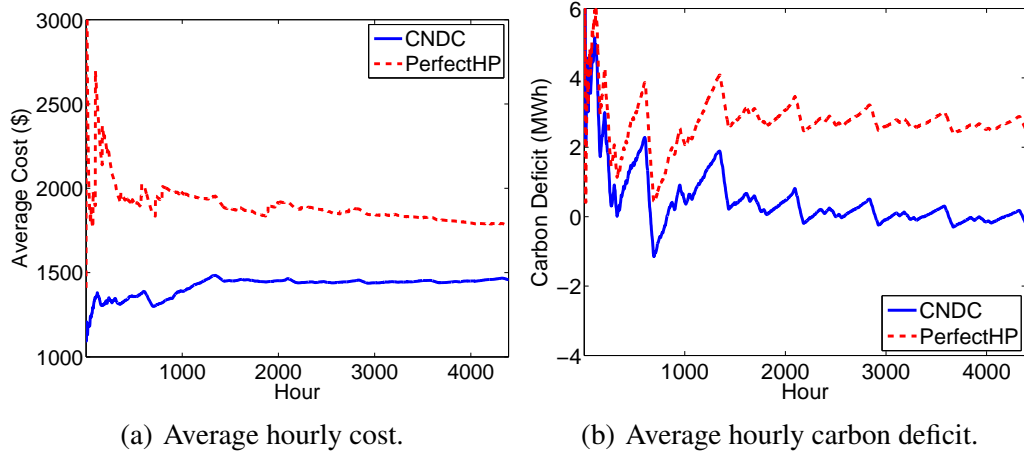


Figure 3.5: Comparison with PerfectHP.

perfectly predicted in advance. Day-ahead prediction of on-site renewable energy is not considered, since it exhibits a high volatility due to its relatively smaller-scale generation. When operating online, the operator minimizes the cost subject to the allocated hourly carbon budget; if no feasible solution exists for a particular hour (e.g., workload spikes), the operator will minimize the cost without considering the hourly carbon budget. We consider day-ahead prediction in the comparison, because prediction beyond 24 hours will typically exhibit large errors [68], especially for solar and wind energy supplies that are commonly used for data centers but heavily depend on weather conditions.

Figs. 3.5(a) and 3.5(b) show the comparison between CNDC and the prediction-based PerfectHP in terms of the average hourly cost and carbon deficit, respectively.<sup>6</sup> The fluctuation of average values is due to the large variation of workloads as well as renewable energy supplies. Figs. 3.5(a) and 3.5(b) demonstrate that CNDC is more cost-effective compared to the prediction-based method with a cost saving of more than 20% over six months. The cost saving mainly comes from the fact that CNDC can focus on cost minimization even though the workload spikes and the carbon neutrality is *temporarily* vi-

<sup>6</sup>The average at time  $t$  in Fig. 3.6 is obtained by summing up all the values from time 0 to time  $t$  and then dividing the sum by  $t + 1$ .

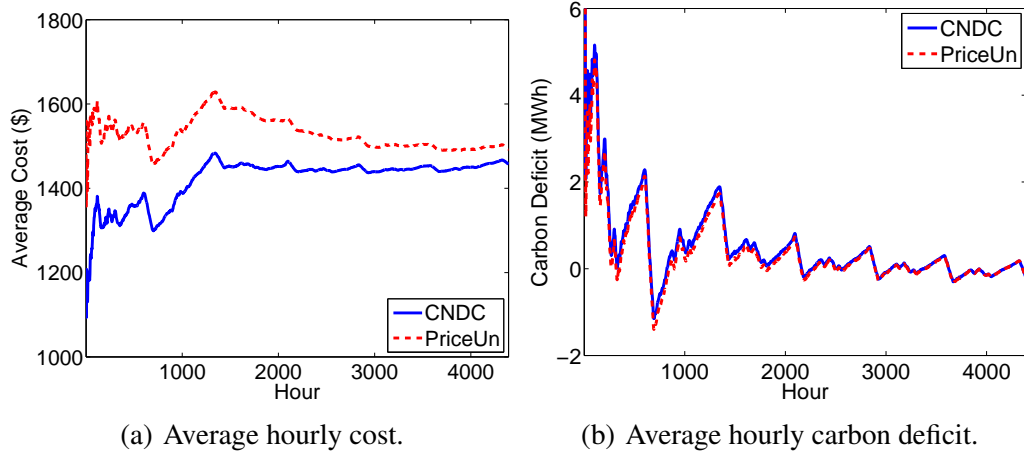


Figure 3.6: Comparison with PriceUn.

olated, since the carbon deficit queue will only penalize the data center for overusing electricity in later time slots while guaranteeing a bounded deviation from the carbon neutrality. By contrast, without foreseeing the long-term future, short-term prediction-based PerfectHP may over-allocate the carbon budget at inappropriate time slots and thus have to set a stringent budget for certain time slots when the workload is high, thereby significantly increasing the delay cost. Note that if day-ahead prediction is leveraged, CNDC can naturally further reduce the cost. In addition to cost saving, CNDC also satisfies the desired carbon neutrality constraint better, as shown in Fig. 3.5(b). Interestingly, Fig. 3.5(b) also shows that prediction-based PerfectHP violates the carbon neutrality constraint. This is because short-term prediction cannot possibly predict the workload spikes in the far future and in such events, more electricity power beyond the carbon budget is required to process the workloads.

#### 3.4.4 Comparison with price unaware methods

We now consider a variant of CNDC but without considering demand-responsive electricity prices. The new algorithm is referred to as PriceUn described as follows.

- **Price-unaware CNDC (PriceUn):** While using the carbon-deficit queue as in CNDC, the data center operator assumes a fixed electricity price (given hourly) without explicitly considering the impact of its energy consumption over the electricity price. At the end of the hour, the data center is charged at a higher price.

Figs. 3.6(a) and 3.6(b) compare CNDC to PriceUn in terms of the average hourly cost and carbon deficit, respectively. The improvement is more than 2.5%, highlighting the importance of incorporating demand-responsive electricity prices in cloud-scale data center operation. While the percentage of cost saving is not too large, it still translates into hundreds of thousands of dollars saving each year. In addition, prior work [107] shows that considering geographical load balancing with demand-responsive electricity prices may further strengthen the cost reduction, pointing to a potential future research direction.

### 3.4.5 Sensitivity Study

This section presents the sensitivity study. We show that CNDC still yields a satisfactory performance in terms of the operational cost while satisfying carbon neutrality: (1) with inaccurate price prediction; (2) with workload prediction errors; and (3) when using measured real server power consumption, workload delay, and toggling energy consumption.

#### **Price prediction error**

The hour-ahead information of non-data-center energy demand may contain errors, affecting the price prediction based on our modeled electricity price function. We include 10% noise in the prediction of non-data-center energy demand when forecasting the hour-ahead demand-responsive electricity price, while everything else is the same as CNDC. We refer to the new algorithm with imperfect knowledge of non-data-center energy demand as RobPrice. Figs. 3.7(a) and 3.7(b) show the comparison of CNDC and RobPrice

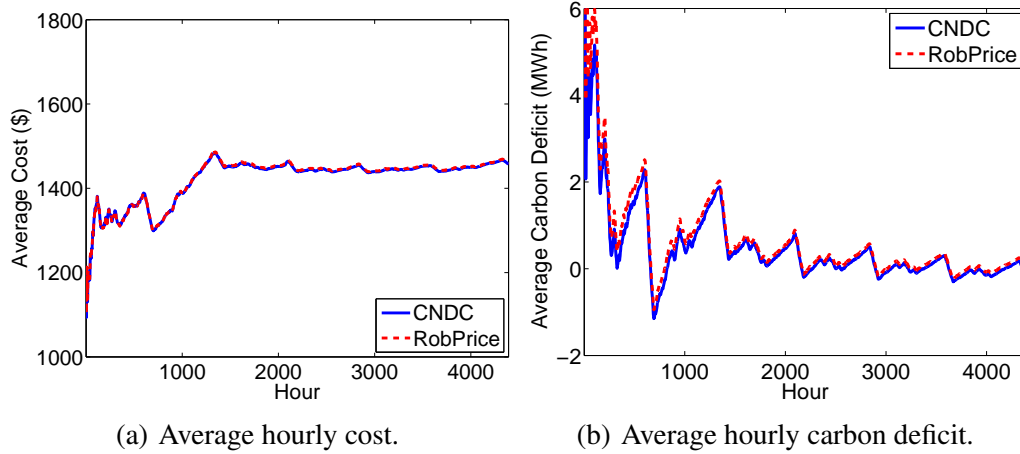


Figure 3.7: Robustness against price prediction errors.

in terms of average hourly cost and carbon deficit, respectively. We see that, even there is 10% error in the knowledge of non-data-center energy demand, the average cost only increases by less than 0.5%, demonstrating the robustness of CNDC against errors in modeling demand-responsive electricity prices. Considering the 2.5% cost saving compared to PriceUn in the absence of price prediction errors, we note that CNDC still outperforms PriceUn even though price prediction errors exist.

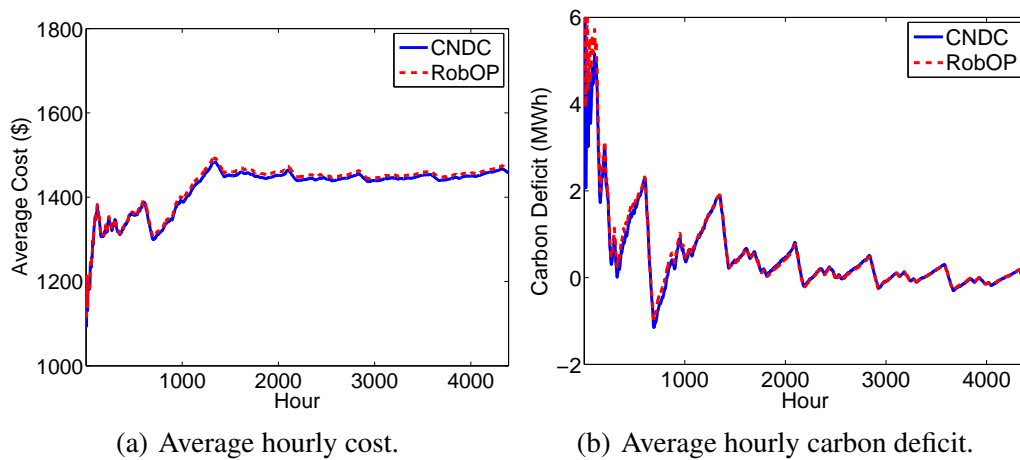


Figure 3.8: Robustness against workload prediction errors.

### **Workload prediction error**

In practice, the hour-ahead workload prediction may not be accurate. To cope with possible workload spikes, we slightly modify CNDC and introduce a new variant, called RobOP, in which the data center operator overestimates workloads by 10% more than the actual values. For both CNDC and RobOP, we choose the carbon-cost parameter  $V$  in such a way that the resulting carbon deficit is zero. It can be seen from Fig. 3.8(a) and Fig. 3.8(b) that the performance of RobOP is quite close to that of CNDC (i.e., approximately 0.5% additional cost on average), demonstrating that CNDC can be successfully applied even though the workloads are conservatively overestimated to accommodate potential traffic spikes. Interestingly, even with workload overestimation, carbon neutrality can still be achieved by RobOP. The reason is that, even though RobOP tends to turn on more servers at the beginning, it will actually turn on fewer servers as the data center operation proceeds, since the carbon deficit queue will pressure the data center operator to use less electricity (which is still sufficient to process the workloads but causes a larger delay cost). Thus, even in the presence of workload overestimation, the carbon deficit queue can still effectively guide the online resource management decisions towards carbon neutrality, while achieving a satisfactory operational cost.

### **3.5 System Experiment**

In this section, we present the experiment study of running CNDC on a real system emulating our considered environment. The experimental results further demonstrate the effectiveness of CNDC in achieving carbon neutrality with satisfactory performance. We first describe our experiment setup and then present the results.

### 3.5.1 Setup

We use a HP Elitepc with a Core i7-3770 CPU (running at 3.4GHz) and 16GB of RAM as the test server. XenServer 6.1 with free license is used to create a virtualized environment with 6 virtual machines (VMs), and each VM is assigned one V-CPU and 512MB of RAM. Each VM emulates a “cluster of physical servers” in our model. In other words, the capacity provisioning knob in our experiment is the number of active VMs: the more VMs, the more power consumption and the better delay performance. Ubuntu Linux server 12.04 32-bit is installed as the VM operating system. Each VM hosts an Apache web server (version 7.0.40), serving a single-tier HTTP web services. Workloads are generated from a separate machine with a Core i7 860@2.8GHz CPU with 4GB of RAM, running Windows 7. The workload generator sends html requests to the VMs. Each html request triggers a random CPU-intensive arithmetic operation embedded in a webpage hosted on the Apache server, where the service time of a single request follows a bounded Pareto distribution (the same as the one used in [95]). In our experiment, we only consider one type of workloads: we use the same I/O trace of MSR [66] that is used in our simulation and scale it to have a maximum VM CPU utilization of 70%. We use network-enabled *Watts Up? .Net* power meter to measure the power consumption. Several trial runs are performed with different workload arrival rates to model the delay and power consumption of the server. Duration of each time slot in our experiment is 1 minute and the total budgeting period is 192 time slots. For factors that cannot be captured by our system (e.g., demand-responsive electricity price, renewable energy), we use real-world trace data as presented in the simulation section.

Note that we subtract the server idle power from the measurements such that the total power consumption is roughly in proportion to the number of active VMs. Although our experiment setup differs from our simulation study and it is a rather small-scale system, it captures the online capacity provisioning problem subject to a long-term constraint with

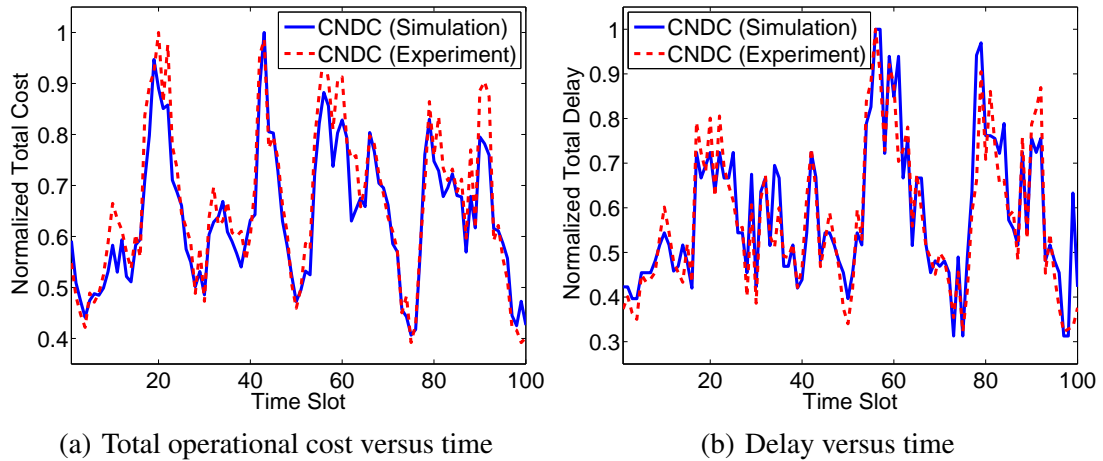


Figure 3.9: Comparison between simulation and experiment results.

demand-responsive electricity prices, which is our main contribution in this work. Moreover, the result is consistent with our analysis, thereby further increasing our confidence in the simulation studies modeling a large data center.

### 3.5.2 Results

In this subsection, we present two sets of results as follows.

#### Instantaneous value

In Figs. 3.9(a) and 3.9(b), we show the instantaneous total operational cost and delay for the first 100 time slots. The reported values are normalized with respect to the maximum value observed in the simulation. We see that the experimental results closely follow the simulation, validating our theoretical analysis and simulation. We believe that the difference between the experiment and simulation is mainly due to measurement errors and the fact that only a finite number of requests are served (which will inevitably introduce errors in terms of the average delay performance). The comparison between the experiment and simulation in terms of the electricity cost is similar and thus omitted for brevity.

### Average value versus $V$

We normalize the reported values with respect to the carbon-unaware algorithm for the convenience of presentation. Figs. 3.10(a) and 3.10(b) show the impact of  $V$  on the average cost (i.e.,  $\bar{g}$ ) and total electricity consumption versus  $V$ , respectively. Under carbon neutrality, the total allowed (normalized) electricity usage is 0.94 compared to that of carbon-unaware algorithm. The carbon-unaware algorithm that minimizes the cost without considering carbon neutrality violates carbon neutrality, whereas CNDC satisfies carbon neutrality by appropriately choosing  $V$  (e.g.,  $V \leq 30$ ). The result conforms with our analysis that with a greater  $V$ , CNDC is less concerned with the carbon deficit while caring more about the cost. This can also be seen from Fig. 3.10(c) and Fig. 3.10(d) that show the average hourly electricity cost and delay cost, respectively: when  $V$  increases, the data center turns on more VMs, leading to a better delay performance while resulting more carbon footprint as well as electricity cost. In the extreme case in which  $V$  goes to infinity, CNDC reduces to the *carbon-unaware* algorithm, which clearly achieves a cost that is a lower bound on the cost that can be possibly achieved by any algorithm satisfying the carbon neutrality constraint. It can be seen from Fig. 3.10(a) and Fig. 3.10(b) that the cost achieved by CNDC is fairly close to the lower bound on the cost achieved by the carbon-unaware algorithm when  $V$  exceeds 30, whereas CNDC satisfies the carbon neutrality constraint for  $V \leq 30$ . This indicates that, with  $V \approx 30$ , CNDC achieves a close-to-minimum cost while still satisfying the carbon neutrality constraint.

Finally, we see from Fig. 3.10 that our experimental result nicely matches the simulation result, validating our simulations and demonstrating the effectiveness of CNDC in real systems.



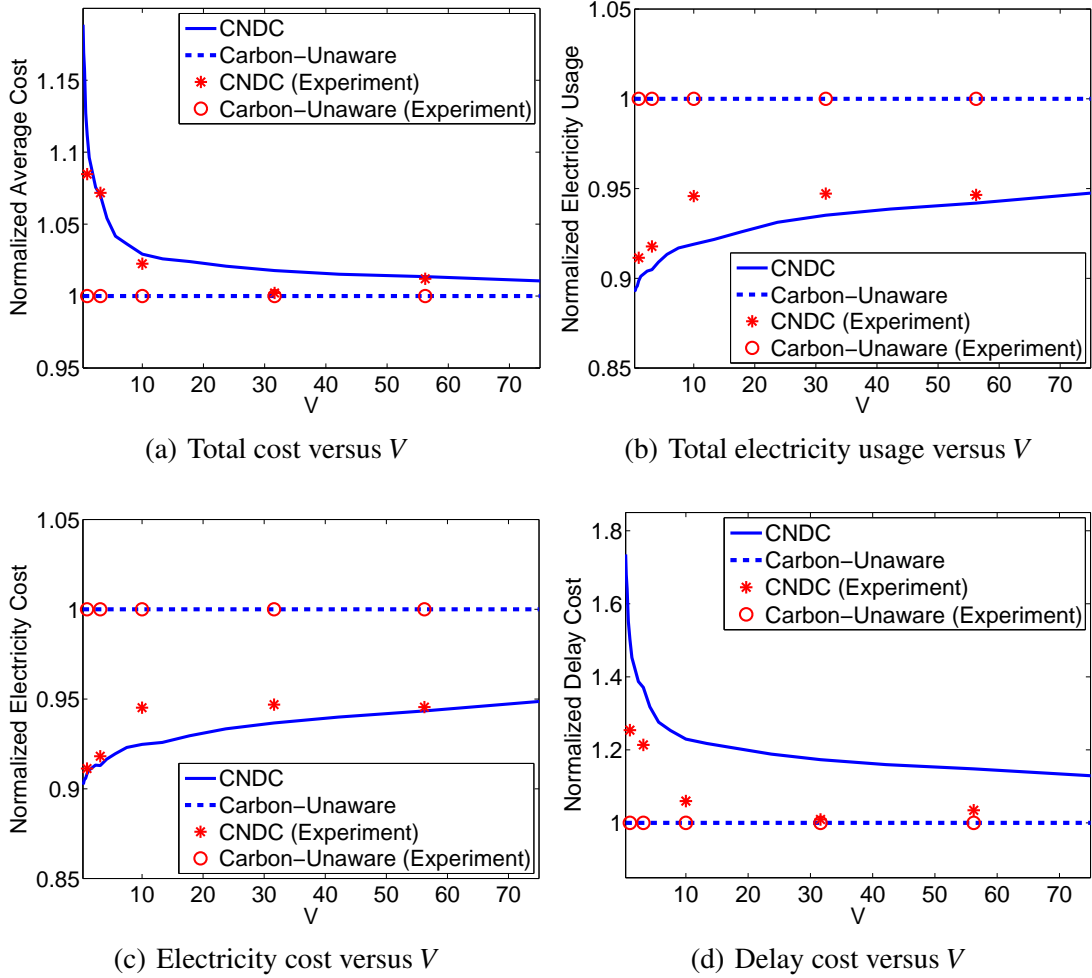


Figure 3.10: Average value versus  $V$ .

### 3.6 Summary

In this part of the dissertation, we proposed a provably-efficient online capacity provisioning algorithm, CNDC, to dynamically control the number of active servers for minimizing the data center operational cost while satisfying carbon neutrality. We explored demand-responsive electricity price enabled by the emerging smart grid technology and demonstrated that it can be incorporated in the data center operation to reduce the operational cost. It was rigorously proved that CNDC achieves a close-to-minimum operational cost compared to the optimal offline algorithm with future information, while bounding the

potential violation of carbon neutrality, in an almost arbitrarily random environment. We also performed trace-based simulation and experiment studies to complement the analysis. The results show that CNDC reduces the cost by more than 20% (compared to state-of-the-art prediction-based algorithm) while resulting in a smaller carbon footprint. In addition, we showed that incorporating demand-responsive electricity price reduces the average operational cost by more than 2.5%. Next, we extend this work for a hybrid data center infrastructure.

## CHAPTER 4

### SUSTAINABILITY IN A HYBRID DATA CENTER INFRASTRUCTURE

While numerous studies have investigated geographic load balancing to minimize carbon emissions of data centers, these studies have primarily focused on self-managed data centers. In this part of the dissertation, we consider a practical hybrid data center infrastructure (including both self-managed and colocation data centers) and propose a novel resource management algorithm based on the alternating direction method of multipliers, called CAGE (Carbon and Cost **A**ware **GE**ographical Job Scheduling) to reduce carbon footprints.

#### 4.1 Background

Reducing an organization's IT carbon footprint heavily depends on its underlying data center architecture. As illustrated in Fig. 4.1, large organizations often use two major types of data centers for housing their servers: self-managed and multi-tenant colocation data centers [12, 55]. Self-managed data centers are owned and operated by the organization. Typically, it requires a huge initial construction cost and dedicated workforce to manage the facility. Meanwhile, to achieve global presence for improving latency performance, these organizations need to deploy servers in many locations where it is not feasible or economical to build and manage an entire data center facility on their own. In such scenarios, colocation data centers play a very crucial role. In a colocation data center, multiple organizations (a.k.a. tenants) rent the space, power and cooling from a third party (colocation operator) who owns/manages the data center facility [55]. Tenants house their servers in designated areas and have full control over their own servers. Unlike self-managed data centers, tenants do not completely control the facility-level power consumption, which depends on the tenants' server power usage as well as the colocation

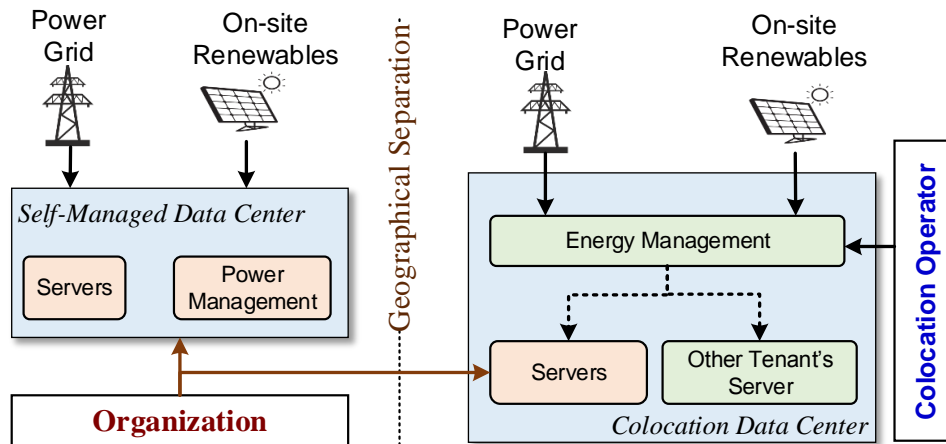


Figure 4.1: Hybrid data center infrastructure.

operator's facility management (e.g., cooling system, renewable generation). A recent study shows that the global colocation market is rapidly expanding, projected to grow to US \$43 billion by 2018 [81]. The combined electricity consumption of colocation data centers is four times higher than that of hyper-scale self-managed data centers altogether (e.g., Apple data center in North Carolina) [88]. Large organizations, like Akamai and even Apple and Amazon, leverage colocation solutions for many of their geographic locations [12, 55].

Although hybrid data center infrastructure is very common in practice and pro-sustainability tenants (e.g., Apple) have already committed to making their servers greener in partnering colocation data centers [12], the existing studies on reducing carbon footprint have primarily been focused on self-managed data centers. For example, numerous studies have leveraged both temporal and/or spatial workload scheduling to minimize the energy consumption, electricity cost, and/or carbon footprint in self-managed data centers [36, 65, 68, 90, 92, 93]. While these studies are promising, they neglect the key difference between self-managed data centers and colocation data centers: in a self-managed data center, all the renewables belong to the organization, whereas in a colocation, the

limited renewables are shared among all the tenants (in proportion to their individual power usage).

**Sharing renewable energy.** In recent years, colocation operators have also been aggressively looking for ways to reduce their carbon footprint by installing on-site renewable energy facilities [12, 119]. In colocations, however, it is not possible to distinguish the source of energy (whether produced from renewables or directly supplied by the utility) because they are distributed by a common power infrastructure. Thus, the renewable energy is considered to be shared among the colocation tenants in proportion to their energy consumption [43, 56]. For example, at a certain time in a colocation with 1MW total IT power consumption and 500KW renewable generation, a tenant that consumes 200KW (20% of the total IT power by all tenants) can only claim 100KW of the renewable energy, whereas the remaining 400KW renewable energy is attributed to the other tenants in that facility. Hence, an organization that does geographical load balancing (GLB) among different geo-distributed data centers for minimizing its carbon footprint should not only consider its self-managed data centers (where all the renewables belong to itself), but also take into account the sharing of renewables in its colocations where it only claims part of the available renewables.

While carbon footprint is an important metric for sustainability, GLB decisions also need to consider electricity cost efficiencies at different locations. Prior studies [36, 68] have shown that there is little correlation between electricity cost and carbon efficiency, resulting in an inherent trade-off between electricity cost and carbon reduction. The trade-off is particularly prominent in the hybrid data center infrastructure: in contrast to self-managed data centers where the electricity consumption from the utility is offset by the on-site renewable, tenants are typically billed based on their actual electricity consumption regardless of the renewable generation in a colocation [28, 55]. Furthermore, the colocation operator may elevate the utility electricity price to account for the cooling

and power infrastructure cost [55], and hence, there is a greater cost impact for increased power consumption in colocation than in the self-managed data center. On the other hand, as described above, to seize a larger portion of the renewable generation in a colocation, the organization needs to increase its servers' power consumption. Therefore, GLB decisions need to incorporate both electricity price for cost saving and carbon footprint for sustainability.

Distributing workloads over a hybrid infrastructure for carbon footprint minimization is challenging, as large organizations (e.g., Google and Microsoft) have hundreds of different interactive services and tens of geographically distributed data centers to process those workloads. A centralized solution suffers from severe performance issues in such scenario. In this part of the dissertation, we propose a distributed resource management algorithm CAGE, based on the *alternating direction method of multipliers* (ADMM), that optimizes GLB decision for a hybrid data center to minimize carbon emission, electricity cost, and revenue loss while meeting performance requirements. CAGE leverages the geographical variation of carbon usage effectiveness (CUE) of electricity production, renewable energy, electricity cost, and cooling efficiency for each data center location. Next, we perform both trace-based simulations and scaled-down prototype experiments to validate CAGE. Our study shows that CAGE can reduce carbon emission by 6.35%, 9.4%, and 37% compared to the algorithm that ignores colocation data centers, minimizes cost, and maximizes performance, respectively.

## 4.2 Model and Problem Formulation

We consider a discrete time model where the organization updates its resource management decisions at the beginning of each time slot.<sup>1</sup> The duration of each time slot may

---

<sup>1</sup>We interchangeably use power and energy, since energy is the product of power and the duration of each time slot.

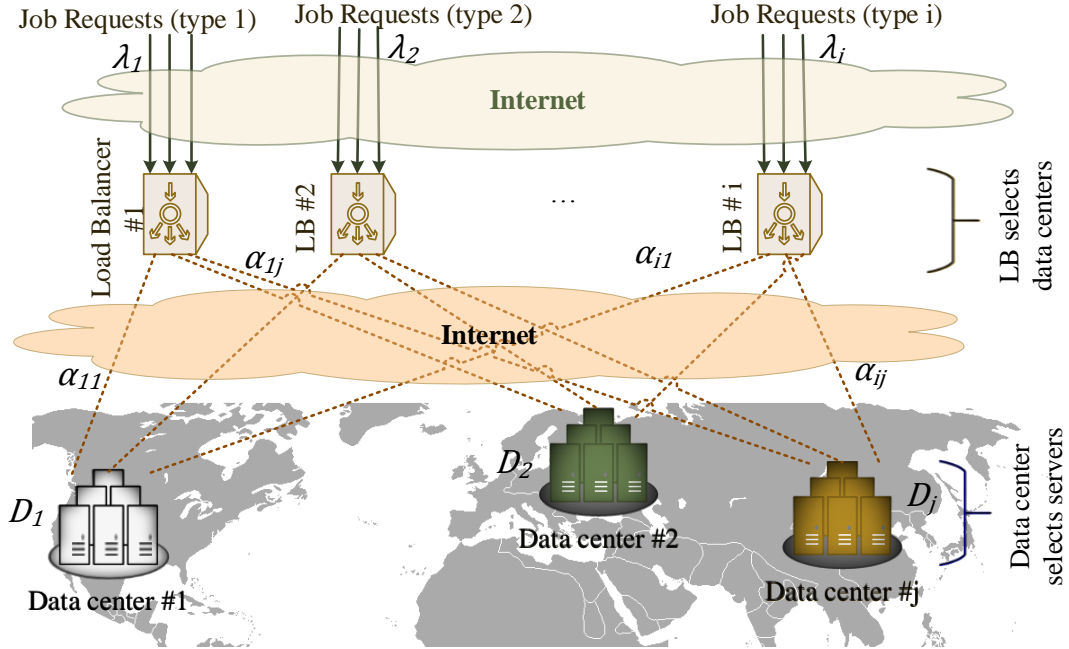


Figure 4.2: Geographically distributed load balancer and data center model. Jobs first arrive at a load balancer, and then distributed to the data centers.

vary from a few minutes to one hour depending on the decision time granularity. We omit the time index in our model for notational convenience.

#### 4.2.1 Data center and propagation delay

We consider an *organization* (e.g., eBay and Facebook) that uses a hybrid data center infrastructure and hosts its servers in  $N$  data centers at geographically distributed locations. The organization has a set of front-end load balancers or gateways  $\mathcal{L}$  where the job requests first arrive and then routed to the data centers for processing. For notational simplicity, we assume that each load balancer  $i \in \mathcal{L}$  routes a specific job type  $\mathcal{J}_i$ . However, a load balancer routing multiple types of job can be viewed as multiple virtual load balancers each routing a single type of jobs. The data centers are indexed by  $j \in \{1, 2, \dots, N\}$ , where  $D_j$  indicates the  $j^{\text{th}}$  data center. We denote by  $\mathcal{D}_{self}$  and  $\mathcal{D}_{colo}$  the set of self-managed and colocation data centers, respectively. In the special case, the

organization uses only self-managed data centers (as studied by prior studies) or colocation data centers. As standard in prior research [68, 92, 93], we focus on delay-sensitive workloads, while delay-tolerant batch workloads are orthogonal to our decisions.

We denote by  $\lambda_i$  the workload (number of job requests) arrival at the  $i^{\text{th}}$  load balancer per unit time. The scheduling algorithm needs to decide the amount of workload to be routed from load balancer  $i$  to data center  $D_j$ , denoted by  $\alpha_{ij}$ . Intuitively, the scheduling decision must satisfy the workload conservation constraint expressed as,  $\lambda_i = \sum_{j=1}^N \alpha_{ij}$ . Each job from load balancer  $i$  has a known average resource demand of  $d_{ik}$ ,  $k \in \mathcal{R}$ , where  $\mathcal{R}$  is the set of all computing resources (e.g., CPU, memory and I/O) [68, 121]. For modeling simplicity, we assume that each data center has homogeneous servers. Nonetheless, a data center which has heterogeneous servers can be added to our model by considering it as multiple virtual data centers each having homogeneous servers. Each server at data center  $D_j$  has a limited set of resources  $S_j = \{s_{jk}\}$  for workload processing, where  $k \in \mathcal{R}$ . The resource constraint for each data center can be expressed as follows:

$$\sum_{i \in \mathcal{L}} \alpha_{ij} \cdot d_{ik} \leq \eta \cdot M_j \cdot s_{jk} \quad \forall j, \forall k \in \mathcal{R}, \quad (4.1)$$

where  $M_j$  is the maximum number of servers available in data center  $D_j$ , and  $\eta \leq 1$  is the maximum utilization factor. As service response delay increases with higher utilization of computing resources [68, 92, 93], the utilization factor  $\eta$  ensures a safety resource margin to compensate for workload and resource demand estimation error.

Latency is arguably the most important performance metric for an interactive cloud service application [36, 68, 121]. A small increase in service latency may cause substantial revenue loss/cost for the organization due to user dissatisfactions. We incorporate revenue cost in our model by mainly focusing on the network propagation latency  $L_{ij}$  between front-end load balancer  $i$  and data centers  $D_j$  while the other latency factors such as queuing and processing delays are essentially captured by Eqn. (4.1) [70, 121]. Following



the existing literature [70, 121], we model the revenue cost (loss due to poor latency) at each front-end load balancer  $i$  as an increasing and convex utility function  $U_i$ , expressed as follows:

$$U_i = \zeta \cdot \lambda_i \cdot \left( \sum_{j=1}^N \frac{\alpha_{ij} \cdot L_{ij}}{\lambda_i} \right)^2 = \sum_{j=1}^N \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i}, \quad (4.2)$$

where  $\sum_{j=1}^N \frac{\alpha_{ij} \cdot L_{ij}}{\lambda_i}$  is the mean propagation latency, and  $\zeta$  is the factor that converts latency into monetary terms.

The background data required to process a specific job type  $\mathcal{J}_i$  may not be available at every data center. Thus, scheduling a specific type of jobs to a data center where data is not available, incurs additional data distribution cost due to the transfer of data from other (nearest) data center. Considering prior studies [37, 121], we model the data distribution cost for job type  $\mathcal{J}_i$  at data center  $D_j$  as follows:

$$b(\alpha_{ij}) = \begin{cases} 0, & \text{if background data required to process job} \\ & \text{type } \mathcal{J}_i \text{ is in data center } D_j, \\ \frac{\zeta \cdot (\alpha_{ij} \cdot F_{ij})^2}{\lambda_i} + H_i \cdot C_{band}, & \text{otherwise,} \end{cases} \quad (4.3)$$

where  $F_{ij}$  is the network propagation latency to bring the data to data center  $D_j$  from nearest place of availability,  $H_i$  is the average data transfer size required for job type  $\mathcal{J}_i$ , and  $C_{band}$  is the bandwidth cost for unit data transfer.

#### 4.2.2 Power consumption and carbon footprints

The major source of data center electricity cost and carbon emission is the energy consumption by the data center while in operation. We consider that the organization turns off idle servers at each location to reduce energy consumption while active (turned on) servers process incoming workloads. Following prior studies, we model the server power consumption  $p_j$  at data center  $D_j$  as an affine function of the average utilization of the

computing resources [66,70,93]. The average utilization of a resource type  $k \in \mathcal{R}$  in data center  $D_j$  can be expressed as follows:

$$\mathcal{U}_{jk}(\alpha_{*j}) = \frac{1}{m_j s_{jk}} \sum_{i \in \mathcal{L}} \alpha_{ij} d_{ik},$$

where  $\alpha_{*j} = \sum_{i \in \mathcal{L}} \alpha_{ij}$  is the number of requests coming to data center  $D_j$ , and  $m_j$  is the number of active servers in  $D_j$ . Thus, we have

$$\begin{aligned} p_j(\alpha_{*j}) &= m_j \cdot (P_I + P_D \cdot \sum_{k \in \mathcal{R}} w_k \mathcal{U}_{jk}(\alpha_{*j})), \\ &= m_j P_I + P_D \cdot \sum_{k \in \mathcal{R}} \frac{w_k}{s_{jk}} \sum_{i \in \mathcal{L}} \alpha_{ij} d_{ik}, \end{aligned} \tag{4.4}$$

where  $P_I$  is the server idle power consumption,  $P_D$  is dynamic power consumption, and  $w_k$  is the weight of resource  $k$  on the power consumption of a server. Using Eqn. (4.1), the minimum number of servers  $m_j$  required for data center  $D_j$  can be computed as  $m_j = \max_{k \in \mathcal{R}} \left( \frac{\sum_{i \in \mathcal{L}} \alpha_{ij} \cdot d_{ik}}{\eta \cdot s_{jk}} \right)$ . Our power model is consistent with existing studies [66, 74, 96], and only requires prior knowledge on per task resource demands, not per task energy consumption. Furthermore, multiplying  $p_j(\alpha_{*j})$  with PUE (power usage effectiveness) factor  $\gamma_j$  yields the data center's total power consumption that includes power consumptions due to cooling, power distribution loss, etc. [74].

To become sustainable, there is a growing trend of using on-site renewable energy (e.g., solar and wind) in both colocation and self-managed data centers. Typically, a data center can be operated partially with such on-site green energy, while consuming the rest of the power from grid. We denote by  $r_j$  the amount of on-site renewable energy available at data center  $D_j$ . In self-managed data centers, the organization can claim all of the renewable energy to offset its carbon footprint. However, in case of a colocation data center, the renewable energy is proportionally distributed among the tenants according to their total power consumption. Thus, in a colocation data center, available renewable

energy share of the organization can be expressed as follows:

$$r_j \cdot \frac{p_j(\alpha_{*j})}{p_j(\alpha_{*j}) + p_{j,other}}, \quad \forall D_j \in \mathcal{D}_{colo} \quad (4.5)$$

where  $p_{j,other}$  is the aggregated power consumption of other tenants at colocation data center  $D_j$ . Thus, data center  $D_j$ 's total amount of power consumption from grid is:

$$e_j(\alpha_{*j}) = \begin{cases} \gamma_j \cdot p_j(\alpha_{*j}) - r_j, & \text{if } D_j \in \mathcal{D}_{self} \\ \gamma_j \cdot p_j(\alpha_{*j}) - r_j \cdot \frac{p_j(\alpha_{*j})}{p_j(\alpha_{*j}) + p_{j,other}}, & \text{if } D_j \in \mathcal{D}_{colo}. \end{cases} \quad (4.6)$$

Note that our study focuses on only *one* organization, while considering the other tenants' power usage as an externally-determined factor independent of our focused tenant's decision. This is reasonable since there are multiple tenants in a colocation and one tenant's decision will not significantly alter the aggregate power of the other tenants. We denote by  $\phi_j$  the amount of carbon emitted for each unit of electricity generation at  $j^{th}$  data center location. Note that,  $\phi_j$  can be obtained by averaging the carbon emission of electricity fuel sources (e.g., oil, and gas) [36]. Thus, the organization's total carbon footprint is  $\sum_{j=1}^N \phi_j \cdot e_j(\alpha_{*j})$ .<sup>2</sup>

### 4.2.3 Electricity Cost

In self-managed data centers, the organization is responsible for setup and maintenance of the facility operation (e.g., cooling, networking, and servers). The organization pays the electricity bill incurred due to the energy consumption from the grid. In colocation data centers, on the other hand, the organization acts as a tenant and only manages its servers while the colocation operator maintains cooling, power distribution, etc. There are two widely adopted pricing models for tenants in a colocation data center: power-based (where tenants are charged based on power subscription regardless of actual energy

---

<sup>2</sup>Carbon footprint of on-site (solar/wind) renewables is typically negligible [36, 68] and hence omitted from our study, but it can be added to our model without affecting our formulation and solution approach.

usage) and energy-based [28,55]. We consider *energy-based* pricing model where a tenant is charged based on its total energy usage (including renewable energy). Total electricity cost at data center  $D_j$  can be expressed as:

$$c_j(\alpha_{*j}) = \begin{cases} u_j \cdot [\gamma_j \cdot p_j(\alpha_{*j}) - r_j]^+ & \text{if } D_j \in \mathcal{D}_{self} \\ u_j \cdot \gamma_j \cdot p_j(\alpha_{*j}) & \text{if } D_j \in \mathcal{D}_{colo}, \end{cases} \quad (4.7)$$

where  $u_j$  is the unit electricity price at the data center's location, and  $[x]^+ = \max(0, x)$ . Note that,  $u_j = 0$  in a colocation means power-based pricing, where tenants pay upfront for their power reservation, and not for their energy usage.

#### 4.2.4 Problem formulation

As revenue and electricity cost are always a major concern for data center operation, it is not practical to aim exclusively at carbon footprint minimization while being unaware of the cost. Hence, we incorporate revenue cost, carbon footprint, and electricity cost into our problem formulation using a parameterized objective function as follows:

$$\text{P21} = \min_{\alpha_{ij}} \sum_{i \in \mathcal{L}} \sum_{j=1}^N \left( \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i} + b(\alpha_{ij}) \right) + \sum_{j=1}^N (\sigma \cdot \phi_j \cdot e_j(\alpha_{*j}) + c_j(\alpha_{*j})), \quad (4.8)$$

where  $\sigma \geq 0$  is set by the organization that determines the relative weight of carbon footprint reduction to cost minimization. A high  $\sigma$  indicates that reducing carbon footprint has more importance than saving cost (electricity and revenue) and vice versa. For very large  $\sigma$ , the optimization objective becomes cost oblivious and only minimizes carbon

footprint. P21 is subject to the following constraints,

$$\lambda_i = \sum_{j=1}^N \alpha_{ij}, \quad \forall i \in \mathcal{L}, \quad (4.9)$$

$$\alpha_{ij} \geq 0, \quad \forall i \in \mathcal{L}, \forall j, \quad (4.10)$$

$$\sum_{i \in \mathcal{L}} \alpha_{ij} \cdot d_{ik} \leq \eta \cdot M_j \cdot s_{jk}, \quad \forall j, \forall k \in \mathcal{R}, \quad (4.11)$$

where (4.9) represents the workload conservation constraint for each load balancer, (4.10) indicates that the amount of workload scheduled to a data center has to be non-negative, and (4.11) ensures that a data center has adequate resources to process scheduled workloads.

### 4.3 Algorithm Design and Analysis

In this section, we first discuss the practical challenges of solving the optimization problem P21 and present background information on the ADMM optimization technique. Then, we show how ADMM can be leveraged to provide an efficient and distributed solution for P21.

#### 4.3.1 Solution to P21

In practice, large organizations (e.g., Google and Microsoft) have hundreds of different interactive services and tens of geographically distributed data centers to process those workloads. Thus, following our discussion in Section 4.2.1, the number of decision variables ( $\alpha_{ij}$ ) for a large organization is in the order of thousands, even more if heterogeneous servers are considered. A centralized solution may suffer from performance issues in such scenario. Dual decomposition can be used to convert P21 into many independent sub-problems for a distributed solution. However, for large-scale problems, such approach often suffers from slow convergence and performance oscillation issues [70].

To avoid such drawbacks, we are motivated to design a distributed algorithm based on the *alternating direction method of multipliers* (ADMM) which we discuss next.

ADMM has received increasing interest recently in solving large-scale distributed convex optimization problems in the areas of machine learning and statistics [18, 70, 111, 121]. It combines the decomposability of dual ascent with the fast convergence of method of multipliers [18]. ADMM is best suited for the problems whose objective function and constraints can be separated into two individual optimization sub-problems with non-overlapping variables. Specifically, given a problem in the form:

$$\min \quad f(x) + g(z), \quad (4.12)$$

$$s.t. \quad Ax + Bz = C, \quad (4.13)$$

$$x \in C_1, z \in C_2, \quad (4.14)$$

with variable  $x \in \mathbb{R}^h$  and  $z \in \mathbb{R}^v$ , where  $A \in \mathbb{R}^{l \times h}$ ,  $B \in \mathbb{R}^{l \times v}$ ,  $C \in \mathbb{R}^l$ , and  $f : \mathbb{R}^h \rightarrow \mathbb{R}$ ,  $g : \mathbb{R}^v \rightarrow \mathbb{R}$  are convex functions. Furthermore,  $C_1$  and  $C_2$  are nonempty polyhedral sets. Thus, the objective function is separable over variables  $x$  and  $z$ , which are coupled through equality constraint (4.13).

Using the method of multipliers [18], the augmented Lagrangian of problem (4.12) can be formed by introducing an additional  $\mathcal{L}2$  norm as follows:

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - C) + (\rho/2)\|Ax + Bz - C\|_2^2, \quad (4.15)$$

where  $\rho > 0$  is the penalty parameter and  $y$  is the Lagrange multiplier for the equality constraint (4.13). Thus, the augmented Lagrangian can be viewed as the unaugmented

Lagrangian with a penalty term  $\rho$ , associated with the problem:

$$\min \quad f(x) + g(z) + (\rho/2)\|Ax + Bz - C\|_2^2, \quad (4.16)$$

$$s.t. \quad (4.13), (4.14).$$

This is clearly equivalent to the original problem (4.12), and hence minimizing  $L_\rho(x, z, y)$  is also equivalent to (4.12). The ADMM algorithm solves the dual problem by updating  $x, z$  and  $y$  at iteration  $t$  as follows:

$$x^t = \arg \min_{x \in C_1} L_\rho(x, z^{t-1}, y^{t-1}), \quad (4.17)$$

$$z^t = \arg \min_{z \in C_2} L_\rho(x^t, z, y^{t-1}), \quad (4.18)$$

$$y^t = y^{t-1} + \rho(Ax^t + Bz^t - C). \quad (4.19)$$

where each optimization step to compute  $x$  and  $z$  can be performed in a distributed fashion. The penalty parameter  $\rho$  acts as the step size for updating the Lagrange multiplier while the iterations continue until convergence.

### 4.3.2 Applying ADMM to solve P21

ADMM cannot be applied directly to solve P21 because its constraints are not separable for each set of variables. Specifically, the load balancer workload conservation constraint (4.9) couples the decision variable  $\alpha$  across data centers while the data center capacity constraint (4.11) couples  $\alpha$  across load balancers. To address this challenge, we reformulate P21 by introducing a set of auxiliary variable  $\beta = \alpha$  as follows:

$$\begin{aligned} \text{P22} = \min_{\alpha_{ij}, \beta_{ij}} \quad & \sum_{i \in \mathcal{L}} \sum_{j=1}^N \left( \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i} + b(\alpha_{ij}) \right) \\ & + \sum_{j=1}^N (\sigma \cdot \phi_j \cdot e_j(\beta_{*j}) + c_j(\beta_{*j})), \quad (4.20) \end{aligned}$$

$$s.t. \quad \lambda_i = \sum_{j=1}^N \alpha_{ij}, \quad \forall i \in \mathcal{L} \quad (4.21)$$

$$\alpha_{ij} = \beta_{ij} \geq 0, \quad \forall i \in \mathcal{L}, \forall j, \quad (4.22)$$

$$\sum_{i \in \mathcal{L}} \beta_{ij} \cdot d_{ik} \leq \eta \cdot M_j \cdot s_{jk}, \quad \forall j, \forall k \in \mathcal{R} \quad (4.23)$$

Clearly, P22 is equivalent to P21 while the formulation of P22 conforms to the ADMM structure described in (4.12). Specifically, considering  $f(\alpha_{ij}) = \sum_{i \in \mathcal{L}} \sum_{j=1}^N \left( \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i} + b(\alpha_{ij}) \right)$ , and  $g(\beta_{*j}) = \sum_{j=1}^N (\sigma \cdot \phi_j \cdot e_j(\beta_{*j}) + c_j(\beta_{*j}))$ , the objective function of P22 is now separable over  $\alpha$  and  $\beta$ .  $\alpha$  determines the revenue cost due to network propagation delay between the load balancers and data centers, and  $\beta$  controls the carbon emission and electricity cost of the data centers. The augmented Lagrangian of P22 can be formed as follows:

$$L_\rho(\alpha, \beta, y) = \sum_{i \in \mathcal{L}} \sum_{j=1}^N \left( \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i} + b(\alpha_{ij}) \right) + \sum_{j=1}^N (\sigma \cdot \phi_j \cdot e_j(\beta_{*j}) + c_j(\beta_{*j})) \\ + \sum_{i \in \mathcal{L}} \sum_{j=1}^N \left( y_{ij} \cdot (\alpha_{ij} - \beta_{ij}) + \frac{\rho}{2} (\alpha_{ij} - \beta_{ij})^2 \right), \quad (4.24)$$

**$\alpha$ -minimization:** According to (4.17), the  $\alpha$  minimization step at iteration  $t$  requires solving the following problem for each load balancer  $i \in \mathcal{L}$ , :

$$\min_{\alpha_{ij}} \quad \sum_{j=1}^N \left( \frac{\zeta \cdot (\alpha_{ij} \cdot L_{ij})^2}{\lambda_i} + b(\alpha_{ij}) \right) + \sum_{j=1}^N \alpha_{ij} \left( y_{ij}^{t-1} + \frac{\rho}{2} (\alpha_{ij} - 2\beta_{ij}^{t-1}) \right), \quad (4.25)$$

$$s.t. \quad \lambda_i = \sum_{j=1}^N \alpha_{ij}, \alpha_{ij} \geq 0, \quad \forall i.$$

The objective function of (4.25) is derived from (4.24) after discarding the irrelevant terms. Thus, the  $\alpha$  minimization step is decomposable over  $|\mathcal{L}|$  sub-problems where each load balancer can solve their own sub-problem independently using standard convex optimization techniques.



---

**Algorithm 2:** CAGE
 

---

**Input:** at the beginning of each decision period, take input carbon efficiency  $\phi_i$ , unit electricity price  $u_i$ , renewable generation  $r_i$  for each data center location. Input workload  $\lambda_i$  for each load balancer and other tenants' power  $p_{i,other}$  for colocation data centers.

**Result:** workload scheduling decision  $\alpha_{i,j}$ ,  $\forall i \in \mathcal{L}, \forall j$ .

- 1 Initialize  $\alpha_{ij}^0 = 0, \beta_{ij}^0 = 0, y_{ij}^0 = 0$ ,  $\forall i \in \mathcal{L}, \forall j$ .
  - 2  **$\alpha$ -minimization:** At  $t^{th}$  iteration, each load balancer solves the sub-problem (4.25) to obtain  $\alpha_{ij}^t$ ,  $\forall i \in \mathcal{L}, \forall j$  and sends it to all the data centers.
  - 3  **$\beta$ -minimization:** Each data center solves the sub-problem (4.26) to obtain  $\beta_{ij}^t$ ,  $\forall i \in \mathcal{L}, \forall j$  and sends it to all the load balancers.
  - 4 Following (4.19), dual variable  $y$  is updated as follows:  
 $y_{ij}^t = y_{ij}^{t-1} + \rho(\alpha_{ij}^t - \beta_{ij}^t)$ ,  $\forall i \in \mathcal{L}, \forall j$ .  
 Updated values are propagated to all load balancers and data centers.
  - 5 Return  $\alpha_{ij}$ ,  $\forall i \in \mathcal{L}, \forall j$  if convergence criteria is satisfied, else begin next iteration by returning to step 2.
- 

**$\beta$ -minimization:** After the  $\alpha$  minimization step, each load balancer propagates corresponding  $\alpha$  values to the data centers. At iteration  $t$ , each data center performs  $\beta$  minimization step by solving the following problem:

$$\begin{aligned} \min_{\beta_{ij}} \quad & (\sigma \cdot \phi_j \cdot e_j(\beta_{*j}) + c_j(\beta_{*j})) + \sum_{i \in \mathcal{L}} \beta_{ij} \left( \frac{\rho}{2} (\beta_{ij} - 2\alpha_{ij}^t) - y_{ij}^{t-1} \right), \quad (4.26) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{L}} \beta_{ij} \cdot d_{ik} \leq \eta \cdot M_j \cdot s_{jk}, \quad \forall k \in \mathcal{R}, \\ & \beta_{ij} \geq 0, \quad \forall i \in \mathcal{L}. \end{aligned}$$

The  $\beta$  minimization step is also decomposable over  $N$  sub-problems where each data center can solve it independently. Finally, the dual variable  $y$  is updated according to (4.19). Key steps and iterations of our proposed distributed solution are summarized in Algorithm 2.

### 4.3.3 Performance of CAGE

Compared to the original problem P21 presented in (4.8), the  $\alpha$  and  $\beta$  minimization steps of CAGE are of a much smaller scale, with only  $N$  and  $|\mathcal{L}|$  decision variables per sub-problem, respectively. CAGE enables each load balancer and data center to solve their own sub-problems independently, providing an efficient and fast distributed solution for carbon awareness in hybrid data center infrastructure. The computational time for a decision in our simulation study (consisting of 8 data centers and thousands of servers) is less than three seconds. We expect similar computational time for even large scale settings since each sub-problem can be solved independently. The optimality and convergence of CAGE can be guaranteed under very mild practical assumptions [18].

**Limitations of CAGE.** One key aspect of the solution provided in CAGE is that while determining the load distribution and resource allocation decisions, the organization needs to know the renewable generation of all the data center locations as well as the power consumption of other tenants  $p_{other}$  at the colocation data centers. While renewable generation at the self-managed data center can be easily measured/predicted for the next decision period [55], the organization does not have a direct approach to acquiring information regarding renewable generation and/or  $p_{other}$  in colocations. However, in today's colocations, the operator can easily communicate the renewable energy information to tenants. Furthermore, the colocation operator can also inform the organization of the total colocation-level power usage, from which the organization can obtain  $p_{other}$  by subtracting its own power usage. This approach eliminates the privacy concerns for other tenants who may not want to share their individual power consumption.

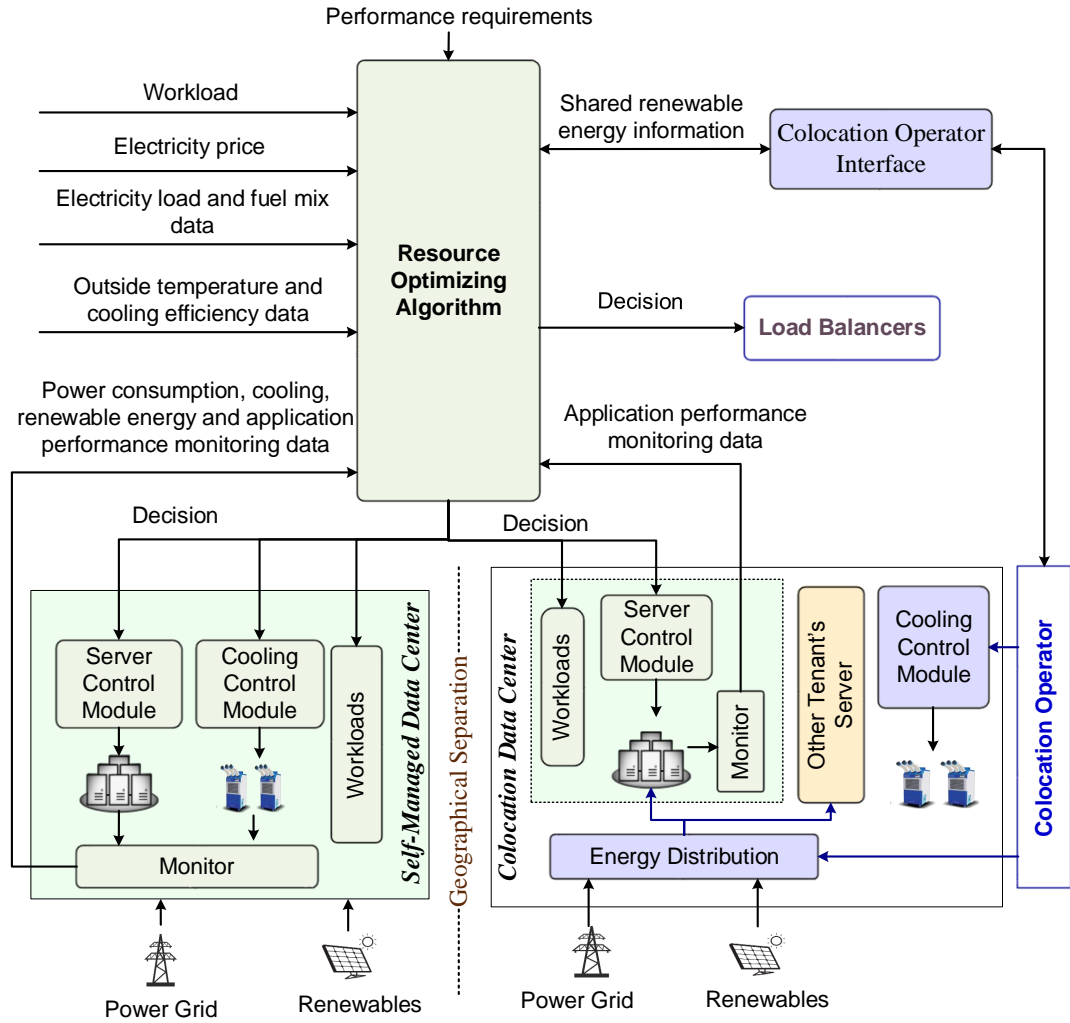


Figure 4.3: CAGE solution overview. This block diagram shows the input, output and the flow of decision for CAGE.

#### 4.4 Simulation Study

In this section, we present our simulation study to evaluate the performance of CAGE. We first describe trace data and simulation settings. Finally, we present benchmark algorithms and compare their performance with CAGE.

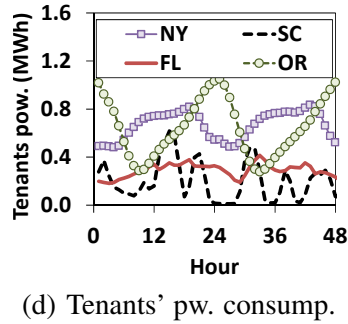
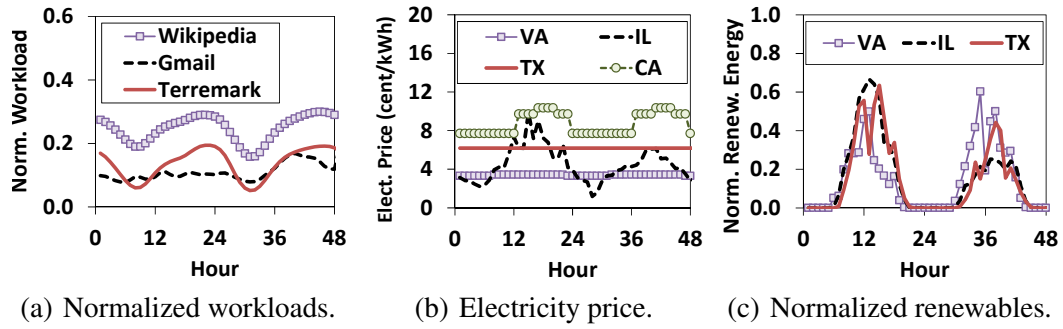


Figure 4.4: Trace data used in simulation and system experiment.

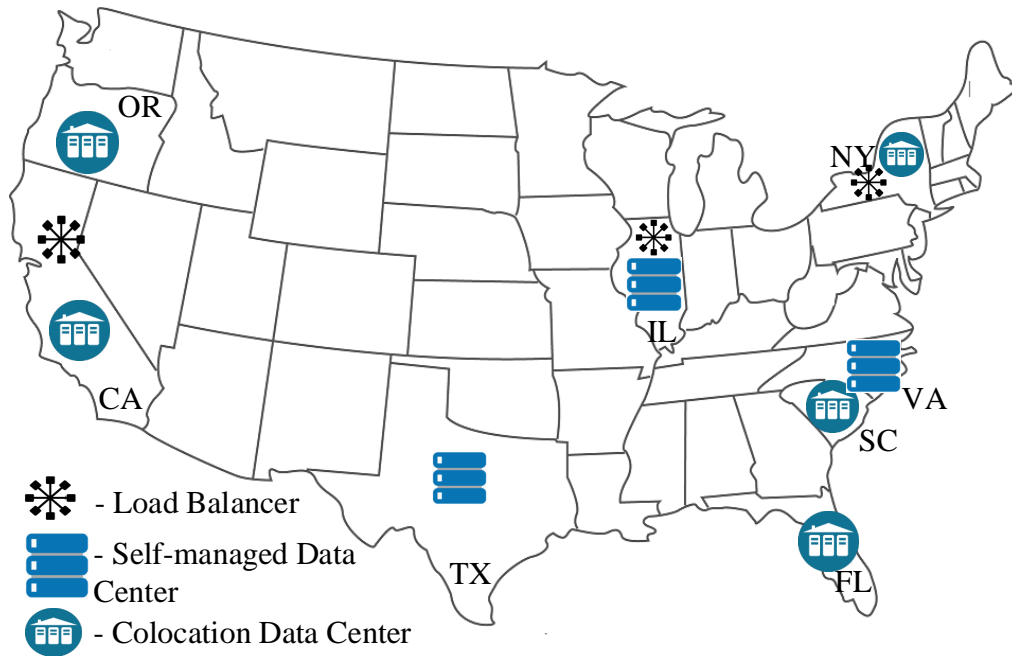


Figure 4.5: Location of the data centers and load balancers.

#### 4.4.1 Data and settings

**Data centers.** We consider 3 load balancers, 3 self-managed and 5 colocation data centers, which are a reasonable fleet of data centers (even for Apple and Facebook) [12]. The locations of the load balancers and data centers are shown in Fig. 4.5. The PUE factor for each data center is randomly set between 1.25 and 1.35 [55, 74]. We consider a network latency of 1 milliseconds per 50 miles distance between a load balancer and a data center [13]. We consider  $|\mathcal{R}| = 3$  computing resources for our simulation: CPU, Memory, and I/O. Following prior studies [15], we set their respective weight ( $w_k$ ) for power consumption as 0.65, 0.15, and 0.2. For each type of resources, the capacity of a server is randomly set between 100 to 150. Note that, all the servers in a specific data center are homogeneous. We have 3 types of jobs where the resource demands for each job type is randomly set between 1 to 5 on each resource dimension. We consider that the data required to process a specific type of jobs is available at randomly selected 4 data centers. The data transfer size is 512KB, 1MB, and 2MB for these job types. We use Wikipedia, Gmail and Verizon Terremark load trace data from [55] as the workload traces for aforementioned 3 job types. We normalize the workload traces with respect to the total capacity of the organization and show in Fig. 4.4(a). As other tenants' energy consumption in the colocation data centers (CA, NY, SC, FL and OR), we use server utilization traces from Microsoft's services Hotmail and MSN, Microsoft Research (MSR), YouTube and FIU university workload traces. Microsoft and Youtube traces are obtained from [55], and the university trace is collected from its web servers. Energy consumption by other tenants at various colocation facility is shown in Fig. 4.4(d). We consider each server has an idle and dynamic server power of 150 Watt and 100 Watt, respectively. The organization's peak IT power rating for self-managed and colocation data centers are 1000kW and 500kW, respectively.

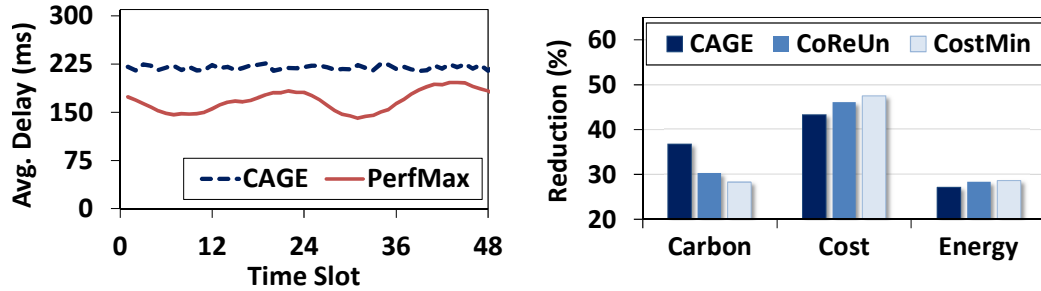
**Renewable energy and carbon footprints.** On-site renewable energy generation is usually proportional to the amount of ambient sunlight available. We obtain the solar radiation at each data center location from [86] for June 2010 and scale the values such that the peak solar energy is 80% of the peak IT power consumption. We show the on-site solar energy generation for some data center locations in Fig. 4.4(c). We skip showing similar trace data figures for others to improve visualization. The carbon footprints of the grid electricity depend on the fuel mix (usage of different fuels) of the electricity generation, often varies over time [36]. We obtain the hourly load and fuel mix data for all the data center locations and derive the aggregated CUE for unit electricity generation. Since more carbon-intensive fuels are used during peak hours, we derive hourly variation of CUE from hourly electricity load data using the method described in [36].

**Other settings.** Electricity prices for the data center locations are taken from respective utility providers. Fig. 4.4(b) shows the unit electricity prices at self-managed data center locations. For colocation data centers, we consider the operator sets electricity price after multiplying it with the PUE and adding 20% to account for infrastructure cost. After an empirical evaluation, the latency-cost parameter  $\zeta$  and the carbon-cost parameter  $\sigma$  are set to  $7.5 \times 10^{-5}$  and 0.02, respectively. These values approximately translate to 50% weight on the carbon emission, 25% weight on latency cost, and 25% weight on electricity cost in optimization decision. The maximum utilization factor  $\eta$  is set to 0.7. The simulation period is 1 week and the duration of each time slot is 15 minutes.

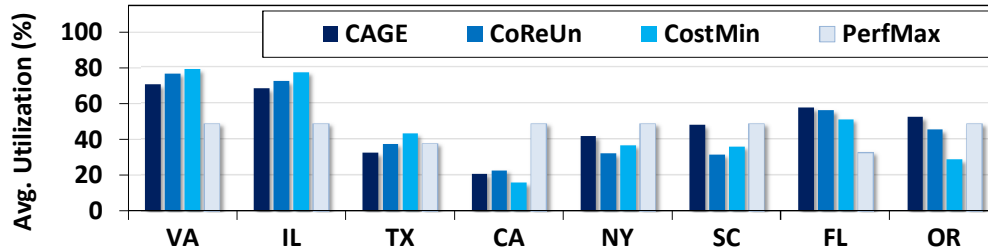
#### 4.4.2 Benchmarks comparison

We first introduce the benchmark algorithms as follow:

- **CoReUn (Colocation Renewable Unaware):** variant of CAGE that does not consider the availability of renewable energy at colocation locations while making workload scheduling decisions. However, the renewable energy is considered in its carbon footprint



(a) Network and processing delays. (b) Reduction compared to PerfMax.



(c) Normalized weekly data center utilization at different locations.

Figure 4.6: Comparison among benchmarks. CAGE achieves 9.4% less carbon emission while incurring 4.1% more in electricity cost compared to CostMin.

after the decision is made. This benchmark represents the current resource management scenarios in sustainable hybrid data center infrastructure.

- **CostMin (Cost Minimization)**: minimizes the electricity and revenue cost while ignoring the organization’s carbon footprint. This benchmark represents a large number of prior studies on self-managed data centers [90, 92, 93, 116, 120].

- **PerfMax (Performance Maximization)**: maximizes the delay performance by keeping all the servers on at every data center location. This is the current practice in most data centers and serves as a benchmark to show how CAGE and other benchmark algorithms are performing in terms of carbon emission reduction compared to an organization whose highest priority goes to the performance [55, 68].

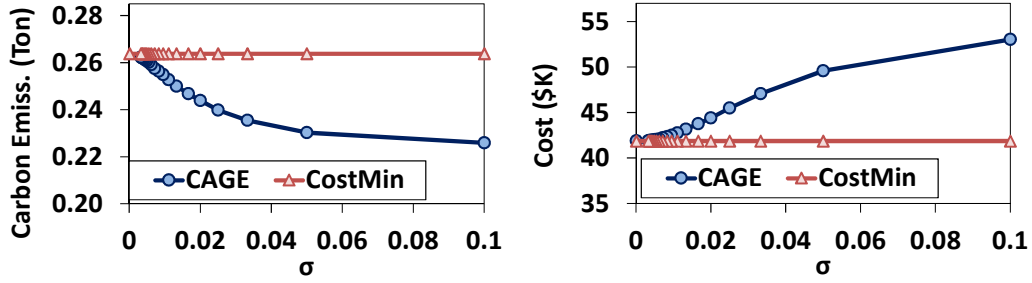
We now compare the performance of CAGE with the benchmarks and show the results in Fig. 4.6. The instantaneous per time slot average delay of CAGE and PerfMax are shown in Fig. 4.6(a). The average delay consists of network propagation latency, job

queuing and job execution delay, as discussed in Section 4.2.1. For clarity, we only show results for the first 48 time slots. As shown in Fig. 4.6(a), the weekly average delay of CAGE is 21% higher than that of PerfMax. CAGE uses as few servers as possible to reduce carbon emission, and hence, the job queuing and processing delays are mostly determined by the maximum utilization factor  $\eta$ . Only network propagation latency varies based on the job scheduling decision. A similar strategy is also used by CostMin and CoReUn to reduce electricity cost and/or carbon footprint. Having almost similar delay performance, these algorithms are omitted from Fig. 4.6(a). On the other hand, PerfMax keeps all the servers on and incurs less delay than any other algorithms at the expense of carbon footprint and electricity cost. Thus, its average delay varies based on the workload amount. The weekly carbon emission, electricity cost and energy consumption of CAGE is 37%, 43%, and 27% lower than that of PerfMax, respectively.

We now compare the electricity cost and carbon emission of CAGE with CoReUn and show the results in 4.6(b). CAGE achieves 6.35% less carbon emission compared to CoReUn while incurring 2.8% more in electricity cost. The carbon footprint reduction comes from the fact that CAGE considers renewable energy availability in colocation locations before making the workload scheduling decision and allocates slightly more workload to colocations (except CA where electricity price is higher) compared to CoReUn. Normalized weekly average utilization of data centers are shown in Fig. 4.6(c). Our separate study with more colocation data centers shows that the improvement in carbon emission becomes significantly higher if the organization has more servers in the colocations than self-managed locations, which is becoming a growing trend in the industry [22].

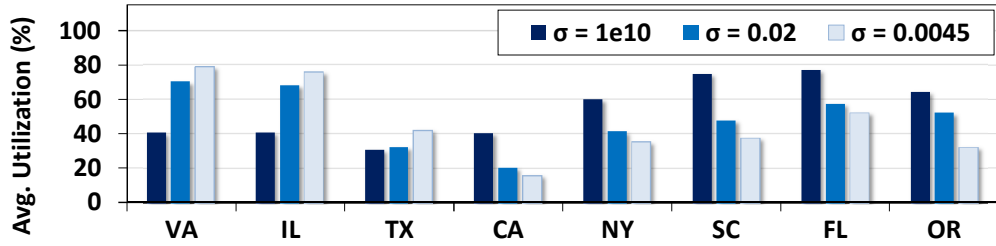
Finally, we compare the electricity cost and carbon footprints of CAGE with CostMin. As shown in Fig. 4.6(c), CostMin dispatches more workloads to the data centers (e.g., VA and IL) where the average electricity price is cheaper. While minimizing the electricity





(a) Total carbon Emission.

(b) Total electricity and revenue cost.



(c) Normalized weekly data center utilization.

Figure 4.7: Impact of carbon-cost parameter  $\sigma$ . For decreasing  $\sigma$ , workloads are shifted from low-carbon (e.g., CA, OR) to low-cost (e.g., VA, IL) data centers.

cost, CostMin ignores the carbon footprint of a data center and assigns more workload to a data center that has lower electricity price and possibly higher CUE factor (e.g., TX). Thus, CostMin becomes a less sustainable solution by focusing only on the electricity cost. The total electricity cost of CostMin is 4.1% less compared to that of CAGE while the carbon emission is 9.4% higher. These results show the benefits of CAGE in terms of carbon emission reduction and highlight the importance of renewable aware workload scheduling in a hybrid data center infrastructure.

#### 4.4.3 Impact of carbon-cost parameter $\sigma$

We now discuss how the performance of CAGE changes based on the carbon-cost parameter  $\sigma$ . As shown in Fig. 4.7(a) and Fig. 4.7(b), when  $\sigma$  is large, CAGE reduces to a carbon usage minimizing algorithm, ignoring the electricity and revenue cost completely.

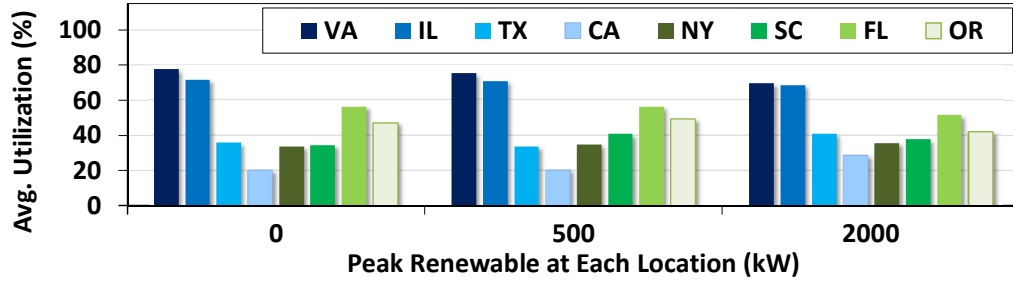


Figure 4.8: Impact of peak renewable energy on CAGE’s weekly workload processing. Workload is shifted from colocation (e.g., FL, OR) to self-managed (e.g., TX) data center with increasing renewable.

When  $\sigma$  approaches infinity, CAGE has minimum carbon footprint and incurs maximum electricity cost. Fig. 4.7(c) shows that the utilization of the data centers at CA, SC and OR are very high for  $\sigma = 1 \times 10^{10}$ . This is mainly because the electricity generation of CA, SC and OR are greener than that of other locations. Hence, CAGE dispatches maximum amount of workload to those data centers to reduce carbon emission. As  $\sigma$  continues to decrease, CAGE becomes cost aware and puts less weight on the carbon emission. When  $\sigma = 0$ , CAGE reduces to CostMin algorithm, minimizing the electricity and revenue cost while ignoring carbon footprints. Specifically, CAGE dispatches more workloads to the data centers (e.g., VA and IL) locations where the average electricity price is lower. Thus, using carbon-cost parameter  $\sigma$ , the organization can balance between carbon footprint and cost to match its policy or desired target.

#### 4.4.4 Renewable energy

In Fig. 4.8 we show the impact of peak renewable energy in the weekly average load distribution among different data center. We see that as peak renewable energy increases at each location, CAGE assigns more workloads to the self-managed data centers. This is mainly because on-site renewable energy is free for the self-managed data centers while the organization has to pay for it at colocation data centers. Moreover, the organization

can claim only a portion (Section 4.2.2) of the renewable energy at colocation locations to offset its carbon emission. Thus, CAGE dispatches more workloads to self-managed locations even if the renewable energy increases at each location equally.

**Other simulation study.** We also perform sensitivity study in terms of workload prediction error and other tenants' power usage estimation error, which are omitted for brevity. In summary, the impact of  $\pm 10\%$ ,  $\pm 20\%$  and  $\pm 30\%$  error in workload prediction are 0.5%, 0.91% and 1.2% difference in carbon emission, respectively, compared to that of perfect workload prediction. The variation in carbon emission are less than 2% for up to  $\pm 30\%$  of other tenants' power usage estimation error, highlighting the robustness of CAGE in wide range of adverse scenario.

## 4.5 System Experiment

In this section, we extend our simulation evaluation by designing and implementing a scale-down test-bed to see how CAGE performs in real life scenario.

### 4.5.1 Test-bed setup

We consider an organization that has two self-managed and one colocation data center. Each self-managed data center has 4 servers while the colocation data center has 8 servers among which 5 belong to two other tenants. We consider that the self-managed data centers are located in VA and IL States, respectively, and the colocation data center is at NY State. We use two load balancers, virtually located at IL and NY state. We use Dell PowerEdge R720 rack servers for each data center. Each server has 6-core Intel Xeon E5-2620 CPU, 32GB RAM, 1 TB hard drive and Ubuntu 12.04 server version as the operating system. The power consumption of each server is measured individually by using CloudPOWER 4x power meter. We implement a data center control software in

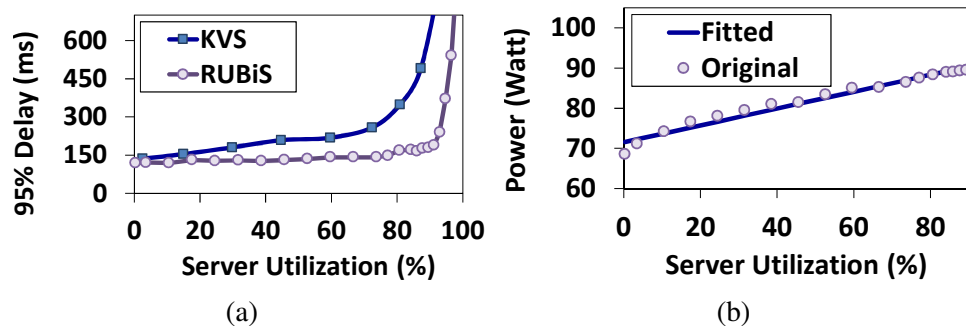


Figure 4.9: Modeling 95th percentile delay and power consumption.

Java that can read the power consumption data from power meters and turn the servers on/off based on the algorithm’s decision.

We consider that the organization is running delay-sensitive RUBiS and key-value-store (KVS) benchmarks [55, 89]. We use Wikipedia and Gmail traces as the workload patterns for these benchmarks. RUBiS is a prototype auction site similar to eBay, implementing the core functionalities such as bidding and buying of an item online. KVS represents the internal processing of multi-tiered social network such as Facebook. The RUBiS and KVS workload generators simulate the online activity of clients (users) where the number of clients indicates the amount of load being generated for the server. We consider 95<sup>th</sup> percentile delay as the performance metric. Fig. 4.9(a) shows how the delay performance of one server changes based on the utilization. Note that, the 95<sup>th</sup> percentile delay of both RUBiS and KVS benchmark increases exponentially as the utilization increases from 70%. This validates our choice of 0.7 for the maximum utilization factor  $\eta$ . We consider CPU as the primary resource for processing the jobs and impacting the power consumption of the test-bed. Following our observation that at 70% utilization, a server can process either 2000 RUBiS client or 35 KVS client, we set the CPU resource demands accordingly for each job type. Now, to model the server power consumption, we first vary the utilization (by varying number of clients from the load

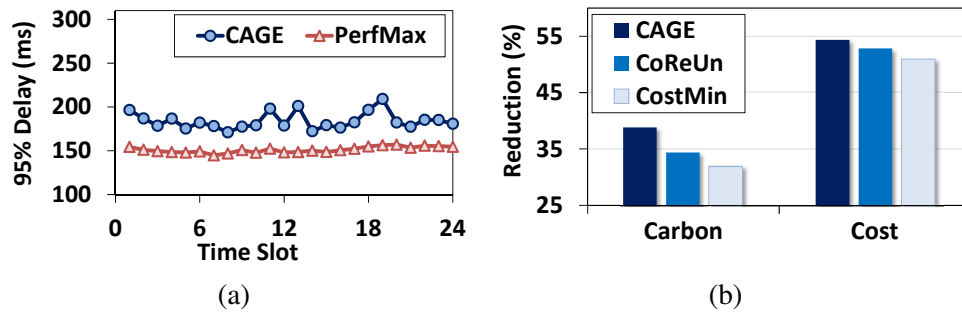


Figure 4.10: Comparison among benchmarks for system experiments. CAGE achieves 4.4% lower carbon emission compared to CostMin, translating to huge potential emission reduction for large organizations.

generator) of a server and obtain corresponding power consumption samples. We use mean squared error fitting on the samples to find the server power model. Fig. 4.9(b) shows both the original power samples and fitted power function which can be expressed as  $p(\text{utilization}) = 71.5 + 21 \times \text{utilization}$ .

#### 4.5.2 Settings and results

We consider 24 time slots and the duration of each time slot is 5 minutes. While making resource provisioning decision, we consider a resource margin of 5% to cope with unexpected system behavior. The combined peak IT power rating of two other tenants is 620 Watt and they are running Httpperf and Hadoop workloads, respectively. Other system experiment settings are kept same as the simulation. We show the system experimental results in Fig. 4.10 which are consistent with our simulation study, and hence we skip detailed discussion. As shown in Fig. 4.10(b), the carbon emission reduction achieved by CAGE, CoReUn and CostMin is 38.7%, 34.3%, and 31.9% compared to that of PerfMax, respectively. Furthermore, Fig. 4.10(b) shows that the electricity cost reduction of CAGE, CoReUn and CostMin is 54.1%, 52.6%, and 50.8% compared to PerfMax, respectively.

In summary, the system experimental results strengthen our observations made by the simulation study and show the effectiveness of CAGE in real life.

#### **4.6 Summary**

In this part of the dissertation, we proposed a novel and distributed geographical workload scheduling algorithm CAGE which reduces the carbon emission of a hybrid data infrastructure consisting of both self-managed and colocation data centers. We leveraged *alternating direction method of multipliers* to design a distributed algorithm and evaluated the performance of our proposed solution by extensive trace-driven simulation studies and real life system level experiments. Next, we address how a cloud service user can take part in the sustainability.

## CHAPTER 5

### SUSTAINABILITY FROM A CLOUD SERVICE USER'S POINT OF VIEW

Many companies (e.g., Microsoft and Amazon) provide data center resources in the form of cloud services to the users. Over the years, cloud services have evolved into various computing models such as Platform as a Service (PaaS) and Infrastructure as a Service (IaaS), which relieves users from the hassle of maintaining their own infrastructure and cater to a broad spectrum of user needs such as scientific computing and web hosting. In many cases, a user can purchase these services online and setup without requiring any extensive technical knowledge about the computing platform. However, a user only manages the acquired resources (e.g., virtual machines) and has no direct control over the corresponding data center energy consumption or carbon emission. An intuitive approach to limit the energy consumption or carbon emission from a user's side is to limit the number of purchased resources. By limiting the spending on cloud resources, a user can contribute to energy consumption capping indirectly and improve environmental sustainability. At the same time, it is also an economical sustainability issue for the user. In this part of the dissertation, we aim to develop a solution for cloud service users to cap their spending on the cloud resources. Since users may have limited technical knowledge, our motivation is to design an autonomous solution that requires less number user inputs.

#### 5.1 Background

Elasticity and scalability are important features of the emerging cloud computing systems, where virtual machine (VM) instances are dynamically purchased/released using autoscaling techniques in an automated fashion. While autoscaling VM instances, cloud users seek two major benefits, i.e., good performance and low expenses. In particular, they often have a cost budget in mind and desire the best performance within their bud-

get. Towards this end, we develop a novel autoscaling algorithm and full-fledged system to optimize delay while satisfying user’s long-term budget constraint (e.g., monthly or yearly budget). Our study focuses on delay-sensitive cloud applications, e.g., web services, for which application performance is measured by the delay of responses.

Supporting budget constraint is essential for common business practice: as shown in a recent survey [103] covering 1,000 data centers by Uptime Institute, over 80% data center operators/managers are given budgets by business departments or higher-level executives at the beginning of a budgeting period (e.g., typically, a month or a year). Such budget constraints are also commonly applied to universities and governments, which typically allocate annual IT operational budgets at the beginning of each fiscal year [105].

Meeting budget constraints while optimizing delay performance is challenging. Re-requesting more VMs at the current time will reduce the available budget for future uses, which may significantly degrade performance and/or exceed budget in the event of high future workloads. Hence, optimally scaling VM acquisitions requires the complete offline information (e.g., workload demand) over the entire budgeting period, which is very difficult, even impossible, to obtain in advance, especially considering highly dynamic workloads and possible traffic spikes due to breaking events [31, 66]. Default autoscaling mechanisms offered by major cloud service providers, such as Amazon EC2 and Windows Azure, typically scale VM instances based on resource utilization indicators such as CPU and memory usage: e.g., adding a new VM instance or switching to a bigger VM instance that has more virtual CPU cores when the current CPU utilization exceeds a certain user-specified threshold [1, 9]. While autoscaling based on resource usage threshold is easy to implement, it cannot optimize the performance while satisfying the budget constraint. It suffers from the following limitations. A too low resource usage threshold may incur an unnecessarily high cost, while a too high threshold reduces the cost, but may result in an intolerable performance. It is difficult to decide optimal resource usage



thresholds *a priori* because their values depend on the user budget and the workloads during the entire budgeting period, but the long-term future workload information is very difficult to predict accurately. Recent efforts on autoscaling for optimizing the performance under long-term budget constraints have primarily focused on evenly distributing budgets across time or predicting the long-term future workloads, neither of which applies to highly-dynamic delay-sensitive workloads in practice [78, 80, 99, 113].

In this part of the dissertation, in view of the practical difficulty in accurately predicting long-term future workloads, we develop an online autoscaling system, called BATS (Budget-constrained AuToScaling), that dynamically scales VM instances to optimize the delay performance while satisfying user budget constraint in the long run. The core of BATS is an online autoscaling algorithm we propose, which only requires the past and instantaneous workload to make effective scaling decisions. The key idea of our algorithm is to keep track of the budget deficit online and incorporate it into the online autoscaling decision: if the actual VM expenses exceed the expected cost thus far, BATS will request fewer and/or smaller VM instances subject to the minimum delay performance requirement, such that the budget deficit can be decreased. Leveraging Lyapunov optimization technique, we formally prove that the BATS produces a close-to-optimal delay performance compared to the optimal algorithm with offline information while satisfying the budget constraint.

As a system, we build a *fully-automated* BATS autoscaler service on Windows Azure. BATS autoscaler only requires user inputs on the desired delay performance and budget of their applications. It manages the performance monitoring, resource planning, and scaling of user applications automatically. We also combine BATS algorithm with a reactive module that monitors runtime performance and handles workload burstiness. We show that the modular design of BATS autoscaler makes its implementation easily adaptable to other cloud platforms such as Amazon EC2.

Table 5.1: List of Notations for BATS Model.

Notation	Description
$\lambda$	Workload arrival rate
$m_j$	# of type- $j$ VM instances
$B$	Total budget
$p_j$	Price for type- $j$ VM instance
$g$	Delay
$d_{\min}$	Minimum (desired) delay
$c$	Cost
$d_{\max}$	Maximum tolerable delay

We evaluate the performance of BATS by running RUBiS benchmark workloads [89] on Windows Azure. The experimental results show that BATS achieves up to 34% less delay compared to the algorithm that evenly divides users budget over all time slots. Compared to the threshold-based scaling rules that are widely used by major cloud service providers, BATS reduces user cost by 10% while achieving a better delay performance. Moreover, the performance of BATS is very close to that of the optimal offline algorithm that knows complete future information.

We also conduct extensive simulation study that complements the system implementation results. We evaluate BATS in terms of both average delay and 95<sup>th</sup> percentile delay, showing the effectiveness of our algorithms on different performance metrics and its scalability on managing applications with hundreds of VMs. We also show that BATS is truly autonomous: it does not need users to select appropriate algorithm parameters or provide additional application information. BATS decides its parameters through online learning and adaptation.

## 5.2 Model and Problem Formulation

### 5.2.1 Model

We consider a discrete-time model by evenly dividing the budgeting period (e.g., typically a month or a year) into  $K$  time slots indexed by  $t = 0, 1, \dots, K - 1$ , each of which has a duration that matches the pricing policy of cloud service providers (CSPs). For example, each time slot can correspond to one hour if we subscribe to Windows Azure or Amazon EC2, both of which charge users for VM instances on an hourly basis. Table 5.1 summarizes the key notations.

- **Autoscaler:** We now present the cloud resource and scaling decision models following the current practice adopted by commercial CSPs. Specifically, there are  $J$  types of VM instances (e.g., small, medium, large) and each type- $j$  VM instance is specified by a set of  $N$  resource configuration parameters  $\mathcal{A}_j = \{a_{j,1}, \dots, a_{j,N}\}$ , where each element represents the provisioning of one resource type. For example, each small-type instance has one virtual CPU and 1.75GB RAM, while each medium one has two virtual CPUs and 3.5GB RAM on Azure [2]. An autoscaler scales VM instances over time. At time  $t$ , the autoscaler requests  $m_j(t)$  type- $j$  VM instances, whose price is  $p_j(t)$ . For notational convenience, we use the vectorial expression  $\mathbf{m}(t) = [m_1(t), m_2(t), \dots, m_J(t)]$  wherever appropriate.<sup>1</sup> Given the autoscaling decision  $\mathbf{m}(t)$ , the cost incurred by the user at time  $t$  is expressed as

$$c(t) = \sum_{j=1}^J [p_j(t) \cdot m_j(t)] \quad (5.1)$$

- **Workloads:** We use “workloads” to represent the demand that needs to be served by the requested VM instances. In our study, we focus on web services and hence, workloads are web requests. We denote by  $\lambda(t) \in [0, \lambda_{\max}]$  the workload arrival rate at time  $t$ , where

---

<sup>1</sup>Our model also supports acquiring VM instances through spot instance market (e.g., offered by Amazon EC2 [10]).

$\lambda_{\max}$  is the maximum possible arrival rate. Arrival rate can be measured by different metrics, such as the number of clients, VM CPU/memory utilization. To quantify the delay performance of web services, our work supports a variety of well-known metrics, such as average delay and tail delay.<sup>2</sup> In our study, we do not restrict the model to any particular delay performance metric. Instead, we use the general notion of  $g(\lambda(t), \mathbf{m}(t))$  to represent the delay performance of interest during time  $t$ , which is jointly determined by the workload arrival rate  $\lambda(t)$  and VM acquisition decisions  $\mathbf{m}(t)$ .

### 5.2.2 Problem Formulation

As workload arrival rate varies over time, the delay performance during high workloads should naturally be given a higher weight than that during low workloads when we evaluate autoscaling algorithms. Hence, our objective is to minimize the long-term delay performance over the entire budgeting period (i.e.,  $K$  time slots), expressed as

$$\bar{g} = \sum_{t=0}^{K-1} \frac{\lambda(t) \cdot g(\lambda(t), \mathbf{m}(t))}{\sum_{t=0}^{K-1} \lambda(t)}, \quad (5.2)$$

where  $g(\lambda(t), \mathbf{m}(t))$  is the delay performance at time  $t$  given the workload arrival rate  $\lambda(t)$  the scaling decision  $\mathbf{m}(t)$ . The term  $\lambda(t)/\sum_{t=0}^{K-1} \lambda(t)$  is a weight that scales the delay at time  $t$  in proportion to the workload arrival rate. There may be a limit on the purchased resources specified by the CSP, which we express as

$$\sum_{j=1}^J a_{j,n} \cdot [m_j(t)] \leq \bar{A}_n, \quad \forall t \text{ and } \forall n = 1, \dots, N, \quad (5.3)$$

where  $a_{j,n}$  is the provisioned resource  $n$  associated with each type- $j$  VM instance and  $\bar{A}_n$  is the limit on resource  $n$ . For example, by default, a maximum of 20 virtual CPUs may be purchased from Azure Cloud Service unless approved for instance increase [9] and, in this case, (5.3) is the constraint on the virtual CPUs.

<sup>2</sup>Tail delay specifies high-percentile latency, e.g., 95th-percentile latency of  $T$  indicates 95% of jobs should have its delay lower than or equal to  $T$ .

The scaling decisions need to satisfy a long-term budget constraint

$$\sum_{t=0}^{K-1} c(t) \leq B, \quad (5.4)$$

where  $c(t)$  is the incurred cost given by (5.1). Note that the budget does not include bandwidth charge, which only depends on the workloads and cannot be *autoscaled*. For each time  $t$ , we also set the maximum and minimum delay constraints, denoted by

$$d_{\min} \leq g(\lambda(t), \mathbf{m}(t)) \leq d_{\max}, \forall t, \quad (5.5)$$

where the maximum delay constraint specifies the worst delay that can be tolerated subject to service level agreement (SLA), while the minimum delay threshold indicates that user experience improvement is negligible by letting the delay go below the threshold [68] and hence there is no need to over-request VM instances.

Note that we can rewrite the delay in (5.2) as  $\bar{g} = \frac{1}{K} \sum_{t=0}^{K-1} \lambda(t) \cdot g(\lambda(t), \mathbf{m}(t)) \cdot \frac{K}{\sum_{t=0}^{K-1} \lambda(t)}$ , where, given workloads and budgeting period, the term  $\frac{K}{\sum_{t=0}^{K-1} \lambda(t)}$  is constant. Hence, we can omit it in the following analysis for notational convenience, and present the (offline) problem formulation for delay minimization as follows:

$$\mathbf{P31} : \quad \min_{\mathcal{M}} \frac{1}{K} \sum_{t=0}^{K-1} \lambda(t) \cdot g(\lambda(t), \mathbf{m}(t)) \quad (5.6)$$

$$s.t., \quad \text{constraints (5.3), (5.4), (5.5),} \quad (5.7)$$

where  $\mathcal{M}$  denotes a sequence of scaling decisions over the budgeting period, which we need to optimize.

### 5.3 Algorithm Design and Analysis

This section presents our autoscaling algorithm, BATS, and analyze its performance. We first present how to the inputs of BATS and then show how to make effective autoscaling

decisions by using the information readily available at the current time step without requiring hardly-accurate long-term prediction. We also formally prove that, for any workload arrivals, the performance of BATS is close to the offline optimal that assumes perfect future information.

### 5.3.1 Obtaining Inputs to BATS

BATS requires two types of inputs:

- **Workload arrival rate:** BATS requires the workload arrival rate  $\lambda(t)$  as its input. In practice, there is a workload predictor that can estimate the instantaneous load, the arrival rate  $\lambda(t)$ , prior to the beginning of time  $t$  using some well-studied learning techniques (e.g., auto-regression analysis) [33, 68]. Note that this prediction is short-term, only for the immediate next time slot, which is different from the long-term prediction of the entire budgeting period required by an offline algorithm. Many prior studies show that such instantaneous workload can often be predicted with a high accuracy [68]. Furthermore, we discuss how to handle unpredictable workload spikes in Section 5.4.2 and quantify the impact of inaccurate prediction in Section 5.6.8.

- **Delay performance:** Delay performance is our optimization objective. In general, the delay increases with increase on arrival rate, decrease on the number or size of VM instances. Nonetheless, the delay performance is also affected by a variety of other factors, such as queuing discipline and load balancing decisions (which may not always be controllable from users' perspective). Thus, it is challenging, even impossible, to mathematically express the delay  $g(\lambda(t), \mathbf{m}(t))$  as an explicit function of  $\lambda(t)$ ,  $\mathbf{m}(t)$ . In practice, we alternatively resort to a delay *lookup* table to empirically measure the delay  $g(\lambda(t), \mathbf{m}(t))$ . Specifically, we create a table whose row and column indexes indicate workload arrival rates and scaling decisions, respectively, and whose entries are the corresponding delay performance. The entries can be populated with some initial estimates (e.g., based on

queueing-theoretic models [91]) at the beginning and then updated in runtime (e.g., using weighted linear regressions) to reflect more accurate delay performance. We discuss how to build such a delay lookup table by offline calibration (Section 5.5.1) and online learning (Section 5.6.5).

### 5.3.2 BATS

Now, we present our online algorithm, BATS, for dynamically autoscaling VM instances. Note first that to optimally autoscale VM instances (i.e., **P31** formulated in Section 5.2.2), complete offline information is required such that the long-term but limited budget can be optimally split across the entire budgeting period, since otherwise performance may be significantly degraded. For example, if more VM instances are requested in the current time slot, less budget is available for future workloads which may potentially increase dramatically. Nonetheless, accurate prediction of such long-term future workload information is quite challenging in practice and sometimes even impossible, considering possible traffic spikes (e.g., due to breaking events) [35, 66]. Hence, autoscaling decisions need to be made online without *a priori* knowledge of long-term future workloads. To circumvent this practical challenge, we leverage the recently-developed sample-path Lyapunov technique [87] to develop BATS.

The key idea is that we make autoscaling decisions using a feedback mechanism to meet the desired long-term budget constraint. Specifically, we use the budget deficit at runtime as the feedback information: if there is a temporary budget deficit (such as due to unexpected high VM costs), we consider both reducing the expenses and managing the delay, so that the budget deficit can be reduced/eliminated eventually. Otherwise, we focus purely on performance optimization.

Mathematically, to track the runtime budget deficit, we construct a (virtual) budget deficit queue which, starting with  $q(0) = 0$ , evolves as follows

$$q(t+1) = \{q(t) + c(t) - z(t)\}^+, \quad (5.8)$$

where  $c(t)$  is the cost at time  $t$ ,  $q(t)$  is the budget deficit queue length and  $z(t)$  is the reference budget for time slot  $t$ . The reference budget  $z(t)$  is not *enforced* as a constraint for the allowed budget over time  $t$ ; instead, it merely indicates how much money we plan to spend for time  $t$ . For example, we can evenly divide the total budget by the total number of budgeting days and obtain daily reference budget, which can be further split to each hour based on the workloads during the prior day. The selection of reference budget  $z(t)$  has a negligible impact on the delay performance under common choices of  $z(t)$ , which we verify by our empirical results in Section 5.6.8.

The budget deficit queue length indicates the deviation of the current cost from the reference budget. Intuitively, with a larger budget deficit at runtime, the autoscaler needs to request fewer VM instances to mitigate the budget deficit at later times for meeting the long-term budget constraint. Thus, the queue length can be leveraged to indicate how much weight we want to give to cost minimization compared to performance optimization, when making autoscaling decisions. To reflect this intuition in our autoscaling decisions, instead of optimizing the delay performance objective in **P31**, we choose to minimize  $V \cdot \lambda(t) \cdot g(\lambda(t), \mathbf{m}(t)) + q(t) \sum_{j=1}^J p_j(t) \cdot m_j(t)$ , where we make autoscaling decisions in an *online* manner based only on the current workload arrival rate, the budget deficit queue length, and a delay-cost parameter  $V$  (which we discuss shortly after). BATS follows the principle **“if exceeding budget, then reduce cost,”** by tracking the budget deficit at runtime and using it as the feedback information to indicate the relative weight/importance of cost minimization versus performance optimization when making autoscaling decisions. The complete description of BATS is provided in Algorithm 1.



---

**Algorithm 3: BATS**

---

**Input:** Input  $\lambda(t)$  (and  $\mathbf{p}(t)$  if applicable), at the beginning of each time slot  $t = 0, 1, \dots, K-1$ .

1 Choose  $\mathbf{m}(t)$  subject to (5.3), (5.4), and (5.5) to minimize

$$\mathbf{P32}: V \cdot \lambda(t) \cdot g(\lambda(t), \mathbf{m}(t)) + q(t) \sum_{j=1}^J p_j(t) \cdot m_j(t) \quad (5.9)$$

2 Update  $q(t)$  according to (5.8).

---

### 5.3.3 Performance of BATS

The following theorem formally shows the performance of BATS.

**Theorem 2.** *Suppose that the instantaneous workload arrival rate and delay performance are perfectly known. Then, for any  $T \in \mathbb{Z}^+$  and  $H \in \mathbb{Z}^+$  such that  $K = HT$ , the following statements hold.*

*a. The budget constraint is approximately satisfied with a bounded deviation:*

$$\frac{1}{K} \sum_{t=0}^{K-1} c(t) \leq \frac{B}{K} + \frac{\sqrt{C(T) + \frac{V}{H} \sum_{h=0}^{H-1} (G_h^* - d_{\min})}}{\sqrt{K}}, \quad (5.10)$$

where  $C(T) = U + D(T-1)$  with  $U$  and  $D$  being finite constants,  $G_h^*$  is the minimum delay achieved over  $t = (h-1)T, \dots, hT-1$  by the optimal offline algorithm with  $T$ -slot lookahead information over  $t = (h-1)T, \dots, hT-1$ , for  $h = 0, 1, \dots, H-1$ , and  $d_{\min}$  is the minimum delay given in (5.5).

*b. The delay  $\bar{g}^*$  achieved by BATS satisfies:*

$$\bar{g}^* \leq \frac{1}{H} \sum_{h=0}^{H-1} G_h^* + \frac{C(T)}{V}. \quad (5.11)$$

*Proof.* Before presenting the proof, we first introduce the benchmark algorithm as follows to compare BATS with. Specifically, we evenly divide the entire budgeting period  $K$  into  $H$  frames where each frame has  $T \geq 1$  time slots. With perfect information over the entire

frame, a  $T$ -step lookahead algorithm solves

$$\mathbf{P33} : \quad \min_{\mathbf{m}(t)} \frac{1}{T} \sum_{t=hT}^{(h+1)T-1} \lambda(t) g(\lambda(t), \mathbf{m}(t)) \quad (5.12)$$

$$s.t., \quad \text{constraints (5.3), (5.4)} \quad (5.13)$$

$$\sum_{t=hT}^{(h+1)T-1} c(t) \leq \sum_{t=hT}^{(h+1)T-1} z(t). \quad (5.14)$$

Essentially, **P33** defines a *family* of offline algorithms parameterized by look-ahead window size  $T$ . In what follows, we assume that for the  $h$ -th frame, where  $h = 0, 1, \dots, H-1$ , there exists at least one sequence of server provisioning decisions that satisfy the constraints of **P33** such that **P33** is solvable. We also specify the minimum value of the objective function in the  $T$ -step lookahead algorithm by  $\frac{1}{H} \sum_{h=0}^{H-1} G_h^*$ , which is the benchmark that we compare BATS with.

Now, we present the following lemma, whose proof can be found in [87].

**Lemma 2.** *For any  $0 \leq \tau \leq K-1$ , any budget deficit queue length  $q(\tau)$ , and any feasible decision, we have*

$$\frac{1}{\tau} \sum_{t=0}^{\tau-1} c(t) \leq \frac{1}{\tau} \sum_{t=0}^{\tau-1} z(t) + \frac{q(\tau)}{\tau}. \quad (5.15)$$

Lemma 2 shows that the budget constraint can be *approximately* satisfied over the  $h$ -th frame if the budget queue length difference  $q(\tau) - q(0) = q(\tau)$  is sufficiently small. Next, we define for notational convenience

$$g(t) = \lambda(t) \cdot g(\lambda(t), \mathbf{m}(t)). \quad (5.16)$$

As an alternative scalar measure of all the queue lengths, we also define the quadratic Lyapunov function  $L(q(t)) \triangleq \frac{1}{2} q^2(t)$ , where the scaling factor  $\frac{1}{2}$  is added for the convenience of analysis. Let  $\Delta_T(t)$  be the  $T$ -slot Lyapunov drift yielded by some control decisions over the interval  $t, t+1, \dots, t+T-1$ :  $\Delta_T(t) \triangleq L(q(t+T)) - L(q(t))$ . Similarly, the 1-slot drift is  $\Delta_1(t) \triangleq L(q(t+1)) - L(q(t))$ . Based on  $q(t+1) = [q(t) + c(t) - z(t)]^+$ ,

it can be shown that  $L(q(t+T)) = \frac{1}{2}q^2(t+1) \leq \frac{1}{2}[q(t) + c(t) - z(t)]^2$ . Then, it can be shown that the 1-slot drift satisfies

$$\Delta_1(t) \leq U + q(t) \cdot [c(t) - z(t)], \quad (5.17)$$

where  $U$  is a constant satisfying  $U \geq \frac{1}{2}[c(t) - z(t)]^2$ , for all  $t = 0, 1, \dots, K-1$ .

Based on the inequality in (5.17), we can easily show

$$\Delta_1(t) + V \cdot g(t) \leq U + V \cdot g(t) + q(t) \cdot [c(t) - z(t)]. \quad (5.18)$$

The online algorithm described in line 2 of Algorithm 1 actually minimizes the upper bound on the 1-slot Lyapunov drift plus a weighted value shown on the right hand side of (5.18). Following (5.18) and considering both BATS and the optimal  $T$ -step lookahead policy (denoted by subscript “ $o$ ”), we can show that, for  $h = 0, 1, \dots, H-1$ , the  $T$ -slot drift plus weighted cost satisfies

$$\Delta_T(hT) + V \sum_{t=hT}^{hT+T-1} g^*(t) \quad (5.19)$$

$$\leq UT + V \sum_{t=hT}^{hT+T-1} g^*(t) + \sum_{t=hT}^{hT+T-1} q(t) \cdot [c(t) - z(t)] \quad (5.20)$$

$$\leq UT + \sum_{t=hT}^{hT+T-1} g_o(t) + \sum_{t=hT}^{hT+T-1} q(t) \cdot [c_o(t) - z(t)] \quad (5.21)$$

$$\leq UT + V \sum_{t=hT}^{hT+T-1} g_o(t) \quad (5.22)$$

$$\begin{aligned} &+ \sum_{t=hT}^{hT+T-1} (t - hT)q^{diff} \cdot |c_o(t) - z(t)| \\ &+ q(hT) \sum_{t=hT}^{hT+T-1} [c_o(t) - z(t)] \\ &\leq UT + VTG_h^* + DT(T-1), \end{aligned} \quad (5.23)$$

where  $g^*(t)$  is the delay achieved by BATS at time  $t$ ,  $G_h^*$  is the minimum delay achieved by the  $T$ -step lookahead policy over the  $h$ -th frame,  $q^{diff} = \max_{t=0,1,\dots,K-1} \{c(t), z(t)\}$  and  $D$

is a finite constant satisfying  $D \geq \frac{1}{2}q^{diff} \cdot \max\{c_o(t), r(t)\}$ . Note that the inequality (5.21) comes from the fact that BATS explicitly minimizes the term  $UT + V \sum_{t=hT}^{hT+T-1} g(t) + \sum_{t=hT}^{hT+T-1} q(t) \cdot [c(t) - z(t)]$  given any  $q(t)$ , and the last inequality follows (5.23) from the constraints in (5.14) satisfied by the optimal  $T$ -slot lookahead policy. Thus, by apply  $q(t) = 0$  and summing up (5.19) over  $h = 0, 1, \dots, H-1$ , we obtain

$$\begin{aligned} q(HT) &\leq \sqrt{UTH + DT(T-1)H + V \sum_{h=0}^{H-1} TG_h^* - V \sum_{t=0}^{K-1} g^*(t)} \\ &\leq \sqrt{UTH + DT(T-1)H + VT \sum_{h=0}^{H-1} (G_h^* - d_{\min})} \\ &= \sqrt{TH} \cdot \sqrt{C(T) + \frac{V}{H} \sum_{h=0}^{H-1} (G_h^* - d_{\min})}, \end{aligned} \quad (5.24)$$

where we define  $C(T) = U + D(T-1)$ . Thus, by recalling  $K = HT$  and Lemma 1, we obtain

$$\frac{1}{K} \sum_{t=0}^{K-1} c(t) \leq \frac{B}{K} + \frac{\sqrt{C(T) + \frac{V}{H} \sum_{h=0}^{H-1} (G_h^* - d_{\min})}}{\sqrt{K}}, \quad (5.25)$$

which proves part (a) of Theorem 2.

Next, by dividing both sides of (5.19) by  $V$  and considering  $q(0) = 0$ , it follows that

$$\sum_{t=hT}^{hT+T-1} g^*(t) \leq \frac{UT}{V_r} + UG_h^* + \frac{DT(T-1)}{V} - \frac{\Delta_T(hT)}{V} \quad (5.26)$$

Finally, by summing (5.26) over  $h = 0, 1, \dots, H-1$  and dividing both sides by  $K = HT$ , we have

$$\bar{g}^* \leq \frac{1}{H} \sum_{h=0}^{H-1} G_h^* + \frac{B + D(T-1)}{V}, \quad (5.27)$$

which shows that the online algorithm can achieve a cost within  $O(1/V)$  to the minimum value achieved by the optimal offline algorithm with  $T$ -step lookahead information. This proves part (b) of Theorem 2. ■

### 5.3.4 Discussion

Theorem 1 provides the *worst-case* performance bound compared to a family of offline algorithms parameterized by their lookahead capabilities characterized by  $T$  (i.e., a larger  $T$  means the lookahead algorithm looks further into the future). The theorem shows that **BATS achieves delay close to offline optimal while approximately satisfying the budget constraint given arbitrary workload arrivals.**

The approximate satisfaction of budget constraint in (5.10) stems from the fact that workloads may be persistently high: budget may be violated in order to satisfy high workloads, even though budget constraint is satisfied in prior time slots. In practice, however, the budget constraint can be strictly satisfied if it is appropriately set (i.e., given workloads, it is feasible to achieve the required delay performance).

**Delay-cost parameter  $V$ .** As formalized in Theorem 1, the delay-cost parameter  $V$  of BATS presents the tradeoff between delay performance optimization and budget satisfaction. When  $V$  becomes larger, BATS tends to minimize the delay, while giving less attention to the incurred cost, because delay carries more weight on the optimization objective (at (5.9)). Thus, an appropriate selection of  $V$  is crucial, but such a value is hard to decide without knowing complete offline information [87]. To address this practical issue, we propose to dynamically update  $V$  as follows:

$$V_{new} = \max\{V_{old} + \beta \times [z(\bar{t}) - c(\bar{t})], V_{\min}\}, \quad (5.28)$$

where  $V_{\min}$  is a sufficiently small positive number to ensure positive  $V_{new}$ ,  $\beta$  is a certain positive scaling factor, and  $c(\bar{t}) = \frac{1}{\bar{t}} \sum_{j=1}^{\bar{t}} c_j(t)$  and  $z(\bar{t}) = \frac{1}{\bar{t}} \sum_{j=1}^{\bar{t}} z_j(t)$  are the cumulative average cost and reference budget per slot up to time  $t$ , respectively. The intuition for using Eqn. 5.28 to update  $V$  is as follows: if there is a budget deficit over quite a few time slots (i.e., cumulative average cost exceeds average reference budget up to time  $t$ ), then we have high confidence that  $V$  needs to be reduced to place more emphasis on cost

minimization such that the long-term budget constraint can be satisfied; and vice versa. Section 5.6.4 shows the desired  $V$  can be dynamically found using 5.28.

**Complexity.** While **P32** in Algorithm 1 involves integer programming (i.e., autoscaling decision  $\mathbf{m}(t)$  can only take integer values), we note that BATS is practically realizable because there are only a reasonably small number of VM types and autoscaling decisions are made only once every time slot. Specifically, given a limit of  $M$  on the number of purchased VM instances and  $J$  types of VM instances, the worst-case complexity is  $M^J$ , which is practically affordable (e.g.,  $M = 20$  and there are four basic types of VM instances in Azure). Moreover, for some applications, there is usually only one type of VM instances that are the most cost-effective (i.e., provides the best performance given the same cost) [79], reducing the exponential complexity to linear complexity. In our experiments in Section 5.5, the computation time of BATS spent on calculating the autoscaling decision is just in the order of milliseconds, while scaling decisions are often made in the order of minutes or a few hours.

### 5.3.5 Reserved Instances

In this section, we discuss how to incorporate reserved instance into our model. Reserved instances typically require a long-term commitment (e.g., one year) that may even exceed users' budgeting period [1, 2]. In other words, reserved instances cannot be easily *autoscaled* in run-time. For completeness, we study how BATS can leverage the discounted pricing of reserved instances although the focus of this chapter is on *autoscaling* on-demand instances due to their elasticity. We present our simulation study by incorporating reserved instances in section 5.6.7. First, we discuss how to modify our model to consider reserved instances, and then how to make VM reservation (purchase) decisions.

### Incorporating reserved instances in model

Despite typically being offered at discounted prices, reserved instances incur upfront reservation fees as well as long-term commitment (e.g., one year). We denote the upfront reservation and fixed usage fees for a type- $j$  VM instance by  $u'_j$  and  $u_j$ , respectively. Following the common practice, we consider that the autoscaler only updates its purchasing decisions for reserved instances every  $K_r$  time slots, which corresponds to the required commitment period. Specifically, the number of reserved type- $j$  VM instances is  $r_j(t)$  at time  $t$ , where  $r_j(t)$  may differ from  $r_j(t-1)$  only when  $t$  is a multiple of  $K_r$ .

To incorporate the reserved instances into our model, we first change the delay notion  $g(\lambda(t), \mathbf{m}(t))$  to  $g(\lambda(t), \mathbf{m}(t), \mathbf{r}(t))$ , the delay during time  $t$ , which is now determined by the workload arrival rate  $\lambda(t)$  as well as VM acquisition decisions  $\mathbf{m}(t)$  and  $\mathbf{r}(t)$ . Next, we include the cost of reserved instances in Eqn. 5.1. Given the autoscaling decision  $\mathbf{m}(t)$  for on-demand instances and  $\mathbf{r}(t)$  for reserved instances, the cost incurred by the user at time  $t$  is expressed as

$$c(t) = \sum_{j=1}^J [p_j(t) \cdot m_j(t) + u_j \cdot r_j(t) + \mathbf{1}_{(t\%K_r=0)} \cdot u'_j \cdot r_j(t)] \quad (5.29)$$

where the first two terms represent costs for on-demand and reserved instances, respectively, the last term is the upfront reservation fee, and the indicator function  $\mathbf{1}_{(t\%K_r=0)}$  is equal to 1 if  $t$  is a multiple of  $K_r$  (i.e.,  $t\%K_r = 0$ ) and 0 otherwise. The limit on the purchased resources specified by Eqn. 5.3, can be expressed as

$$\sum_{j=1}^J a_{j,n} \cdot [m_j(t) + r_j(t)] \leq \bar{A}_n, \quad \forall t \text{ and } \forall n = 1, \dots, N, \quad (5.30)$$

where  $a_{j,n}$  is the provisioned resource  $n$  associated with each type- $j$  VM instance and  $\bar{A}_n$  is the limit on resource  $n$ .

Decisions for the reserved instances require commitment over  $K_r$  periods and thus satisfy

$$r_j(t) = r_j(t+1), \forall (t+1) \% K_r \neq 0 \text{ and } \forall j = 1, \dots, J. \quad (5.31)$$

With the changes discussed above, BATS can incorporate reserved instances into its operation.

### **Purchasing reserved instances**

While reserved instances cannot be *autoscaled* and hence are not our focus, we discuss how to make VM reservation decisions for the completeness of study.

Although workloads vary over time, there is often a minimum *baseline* workloads: workload arrival rates rarely fall below the baseline. Thus, one can leverage discounted prices by purchasing reserved instances to serve the baseline workloads, which are often time-invariant. Nonetheless, optimally deciding the reservation of VM instances is challenging: it involves jointly optimizing the decisions for both reserved and on-demand instances as well as predicting the future workloads over a long timescale.

There have been some *theoretical* studies, e.g., [108], which jointly optimize the purchasing decisions for reserved and on-demand instances. In these studies, a key assumption that may not hold in practice is that users stay in the cloud market for a sufficiently long period, compared to which the commitment period for reserved instances is rather short [108]. However, in practice, users often have a monthly or yearly budget, whereas the commitment period of reserved instances is typically one year or even longer [10]. Thus, purchasing reservation instances is a longer-term decision, compared to the timescale of our interest (i.e., satisfying the user's monthly or yearly budget). In other words, optimally purchasing reservation instances needs to be pre-determined and is



in parallel to our focus.<sup>3</sup> Hence, we assume that the reservation instances are exogenously given and may be determined using the following approach. Before using cloud services, the users first make a projection of its long-term workloads based on its past workload history and/or its expected workload growth over the required commitment period. Then, the user decides the minimum (expected) workload as its baseline and reserves VM instances accordingly.

While clearly the above approach to purchasing reserved instances may not be optimal, it matches the current practice that users' budgeting periods are often shorter than or equal to the commitment period required by the CSP. Hence, in this dissertation, we leave the purchasing decisions for reserved instances as orthogonal to our focus of *autoscaling* on-demand instances subject to the user's budget constraint.

#### 5.4 System Implementation

For evaluation, we build a fully-automated BATS autoscaling service on Windows Azure. Our service allows users to autoscale VMs running their Azure applications, conveniently, effectively and reliably. It provides a graphical user interface (GUI) for users to provide the cost and performance requirements of their applications. Users specify the budgeting period and the total budget as their cost requirement, and they also specify their performance requirement: the maximum tolerable application delay  $d_{max}$  and desired delay  $d_{min}$ , as illustrated at Equation 5.5. Note that if inappropriately set (e.g., budget is too small), the budget and the maximum tolerable delay may not be possibly achievable at the same time. One approach to avoid such inappropriate settings is that BATS provides some general guidance to cloud users based on history data before BATS starts running. Specifically, the minimum required budget can be calculated at the beginning of a bud-

---

<sup>3</sup>Some CSPs such as Azure Cloud Service currently do not support combination of reserved and on-demand instances [9].

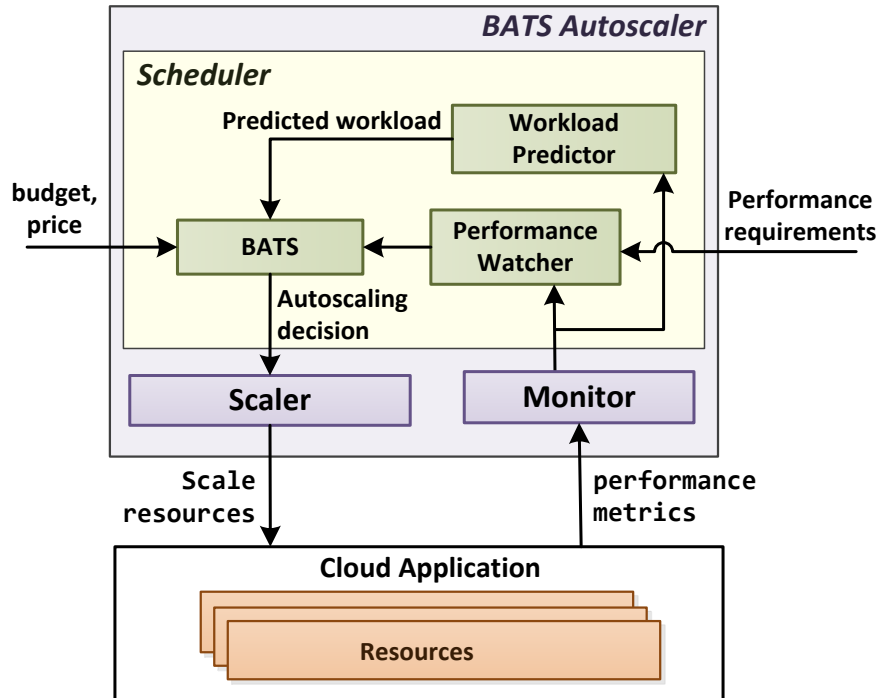


Figure 5.1: Architecture diagram of BATS autoscaler.

getting period based on the previous workloads (plus a certain margin) and the tolerable delay performance given by the user. A warning will be prompted if the user provides a budget below the required budget. Using a similar principle, run-time warning can also be activated, if the current budget deficit (or money spent so far) is too large. The GUI also requests application information that grants the autoscaler to access the management portal provided by the CSP, such as what metrics to be used for indicating the workload arrival rate, and where to store/fetch the performance data. Once BATS is configured, it autonomously monitors user application and dynamically scales VM resources using BATS algorithm. It also reacts to runtime performance feedback, improving system robustness on handling workload prediction inaccuracy and burstiness. This section presents the system implementation of BATS autoscaler service.

### 5.4.1 Architecture Overview

Fig. 5.1 presents the software architecture of BATS, consisting of three main modules — *Monitor*, *Scheduler* and *Scaler*.

*Monitor* gathers different performance metrics of the hosted cloud application and provides the data to *Scheduler*. In Azure, a cloud application writes its performance metric values to a specified Azure table storage periodically, which *Monitor* accesses them to collect performance information of the application. To model performance of different applications, *Monitor* supports using different performance metrics, such as CPU usage, memory usage, and the number of web server connections. For example, for a CPU-intensive application, monitoring CPU usage could be more appropriate than monitoring network traffic. BATS allows users to select performance metrics suitable for their applications.

*Scheduler* decides the optimal number of VMs required by the cloud application based on BATS algorithm with a goal of minimizing delay while meeting the budget constraint. *Scheduler* is the core of BATS autoscaler service, which we describe in more details in Section 5.4.2.

*Scaler* executes the scheduler's decision and submits the scaling request to the cloud. It handles the underlying details of connecting to the cloud, certificate management and service status information retrieval. For example, Azure provides a service management API to control the cloud resource configuration. To use this API, the users need to create a management certificate in the cloud portal and import it through the GUI of BATS. Whenever *Scheduler* issues a scaling decision, *Scaler* creates and uploads the new VM configuration using the management certificate. Then, Azure scales the VM instances for the hosted application accordingly.

We separate *Monitor* and *Scaler* from *Scheduler* to improve modularity: *Monitor* and *Scaler* require usage of cloud-specific APIs, while *Scheduler* is cloud-independent.

This design allows BATS to support other CSPs with small modifications to *Monitor* and *Scaler* only. For example, to apply BATS on Amazon EC2, *Scaler* connects to Amazon EC2 using AWS credentials; *Monitor* periodically obtains performance metric values using the APIs provided by Amazon EC2, i.e., CloudWatch service.

### 5.4.2 Scheduler

*Scheduler* is the core of our autoscaler service, which consists of three submodules — *performance watcher*, *workload predictor* and *BATS algorithm*. *Scheduler* operates in both proactive and reactive manner. Its proactive behavior is implemented at BATS algorithm, which determines the VM scaling decision at the beginning of each time slot by considering the estimated workload given by workload predictor. The reactive behavior takes runtime performance feedback into consideration that is implemented by performance watcher, handling workload prediction inaccuracy and burstiness.

**Workload predictor** predicts the upcoming workload by analyzing the past values. As a key advantage, BATS does not require long-term workload prediction, which is hardly accurate in practice. Instead, BATS only needs workload prediction for the next time slot. Since the prediction model is not a contribution of this dissertation and many other prediction models exist [30,38,39,115], we choose and implement an auto-regressive model shown in [67]. This model predicts  $f(d, t)$ , the value of a chosen performance metric at time  $t$  of day  $d$ , by taking the moving average of the previous  $N$  days at the same time  $t$ . Mathematically, the predicted value of a performance metric  $f$  at time  $t$  is given by  $f(d, t) = \frac{1}{N} \sum_{i=1}^N a_i * f(d - i, t) + c$ . The parameters  $a_i$  and  $c$  are calibrated online using history data. We can also integrate other prediction algorithms into BATS, and as discussed next, performance watcher helps us to readjust the scaling decision in the event of prediction error.

**Performance watcher** monitors two types of events continuously: (1) the current workload arrival and (2) the delay status. If there is a significant difference between the predicted workload and observed workload or if the maximum delay cap  $d_{max}$  is violated, performance watcher triggers BATS algorithm module to recalculate the scaling decision for the current time slot. Performance watcher is designed to avoid persistent overloading and recover from inaccurate prediction and/or bursty workloads reactively.

**BATS algorithm** implements BATS algorithm presented in Section 5.3, which determines the VM scaling decision based on the predicted workload arrival before each time slot begins. As it takes about 10 minutes to acquire new VMs in Azure, BATS submits the proactive scaling decision 10 minutes before each time slot begins. In addition, it takes runtime feedback from performance watcher and recalculates the desired VM scaling decision in the event of mis-predicted and/or bursty workloads.

## 5.5 System Experiment

This section presents experimental results of using BATS to autoscale VM instances hosting a RUBiS web application on Windows Azure. We first describe the application, the cloud service and our experimental setup. Then, we compare BATS with other benchmarks. Our results show that BATS achieves up to 34% less average delay compared to the algorithm that evenly divides users budget over all time slots. BATS also reduces users cost by 10% while achieving less delay compared to widely-used reactive scaling rules. Moreover, the performance of BATS is very close to that of the optimal offline algorithm that knows complete future information.

### 5.5.1 Experimental Setup

We deploy RUBiS web application, which implements the core functionality of an auction site: selling, browsing, and bidding. RUBiS is widely used to evaluate the perfor-

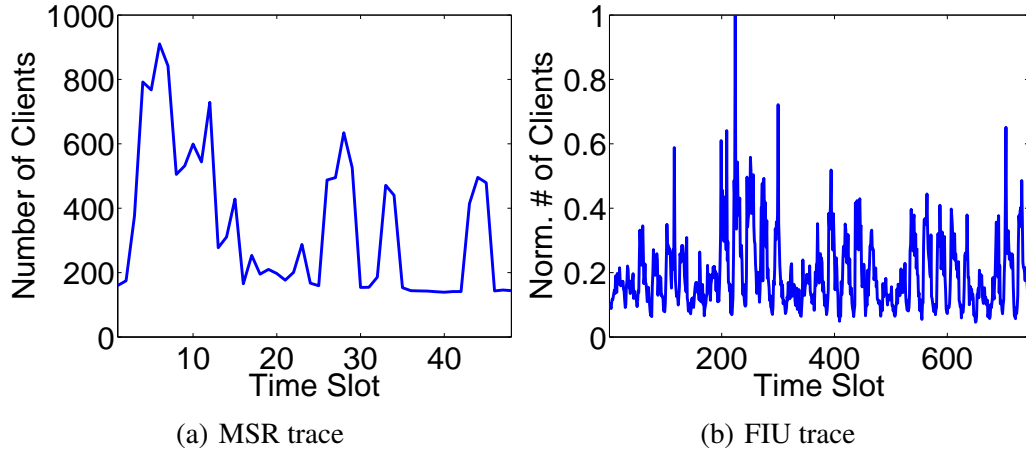


Figure 5.2: Workload traces.

mance and scalability of application servers and virtualized environments [89]. It follows a three-tier web architecture: a front-end web server tier, business logic tier and back-end database tier. We run RUBiS on Windows Azure Cloud Services, which provides a Platform-as-a-Service (PaaS) environment for hosting multi-tier scalable web applications [2].

**Workload.** We use RUBiS workload generator to send client requests to the cloud-end servers. The workload generator creates user sessions (a.k.a clients) which simulates the browsing of an auction site like eBay. The number of clients indicates the amount of workload being generated for the target web site. The average execution time for each web page varies based on the underlying computation. We generate workload arrivals (Fig. 5.2(a)) based on a workload trace extensively used in prior work (e.g., [66]), representing the activity trace of a few thousand users on enterprise file servers at Microsoft Research.

**System configurations.** We deploy the three-tier RUBiS application on Azure Cloud Services. Our experiments scale the VM instances in the range of 1 to 20, where 20 is the default limit set by Azure [9]. We choose to use extra-small VMs only because they offer the most cost-effective way to execute RUBiS workload. For example, the price of

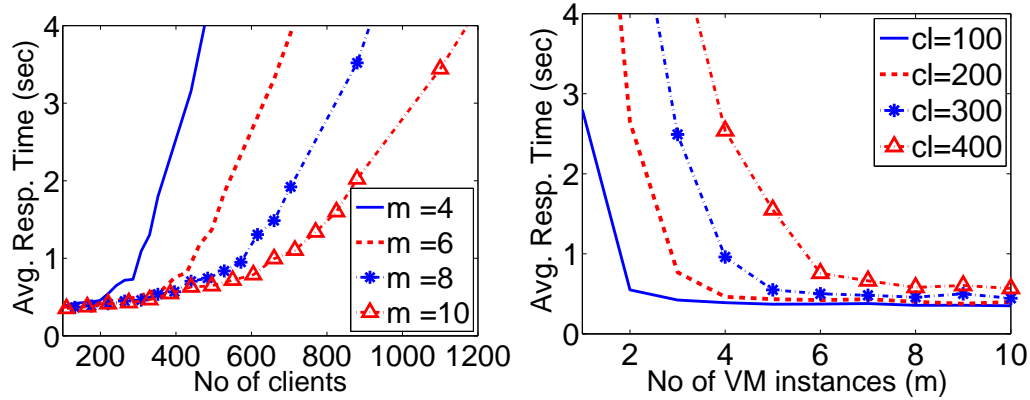


Figure 5.3: RUBiS load response time correlation.  $m$  and  $cl$  denote the number of VM instances and clients respectively.

a small VM is 4 times of the extra-small VM, but its throughput is only 3 times, while the larger VMs are even worse in terms of the cost effectiveness. To avoid excessively long experimentation time, the budgeting period in our study consists of 48 time slots and the duration of each time slot is 1 hour. The default budget for our experiment is \$8.5 while the cost for one VM instance per hour is \$0.02. In our experiment, we set the desired average delay  $d_{min} = 520$  ms, and the maximum tolerable average delay  $d_{max} = 1500$  ms. The delay settings for our RUBiS application are consistent with prior studies [51, 122].

**Autoscaler inputs.** To enable autoscaling for RUBiS, our autoscaler uses the number of web connections to monitor incoming workload. We model the load delay correlation of RUBiS workload using a delay lookup table, which is built at calibration phase by varying the number of clients from the workload generator and obtaining delay for each different scaling configuration (i.e., number of VM instances). We discuss how to build the delay lookup table online in Section 5.6.5. Fig. 5.3(a) shows the average delay under different numbers of clients: average delay increases slowly up to certain load, and then it increases exponentially at heavy load (i.e., saturated). Fig. 5.3(b) shows that if we increase the number of VM instances, average delay decreases down to around 400 ms,

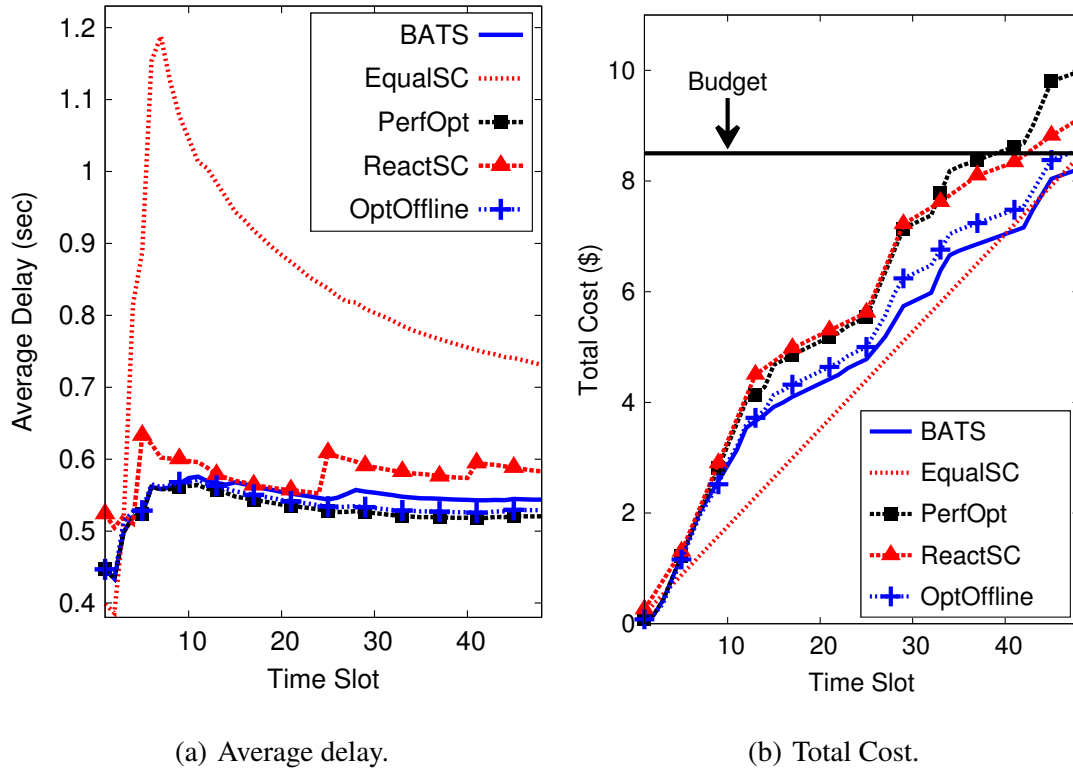


Figure 5.4: Comparing BATS with other algorithms.

after which it becomes almost constant as the average delay is dominated by the request execution time and there is no further delay reduction even if we add more VMs.

### 5.5.2 Experimental Results

We conduct three sets of experiments: (1) compare BATS with three well-known autoscaling algorithms and offline optimal; (2) show the impact of user budget on the performance of BATS; and (3) show the delay-cost tradeoff.

#### Comparison with other autoscaling algorithms

We compare the performance of BATS with three online autoscaling algorithms and the optimal offline algorithm. We first describe the benchmark algorithms as follow:



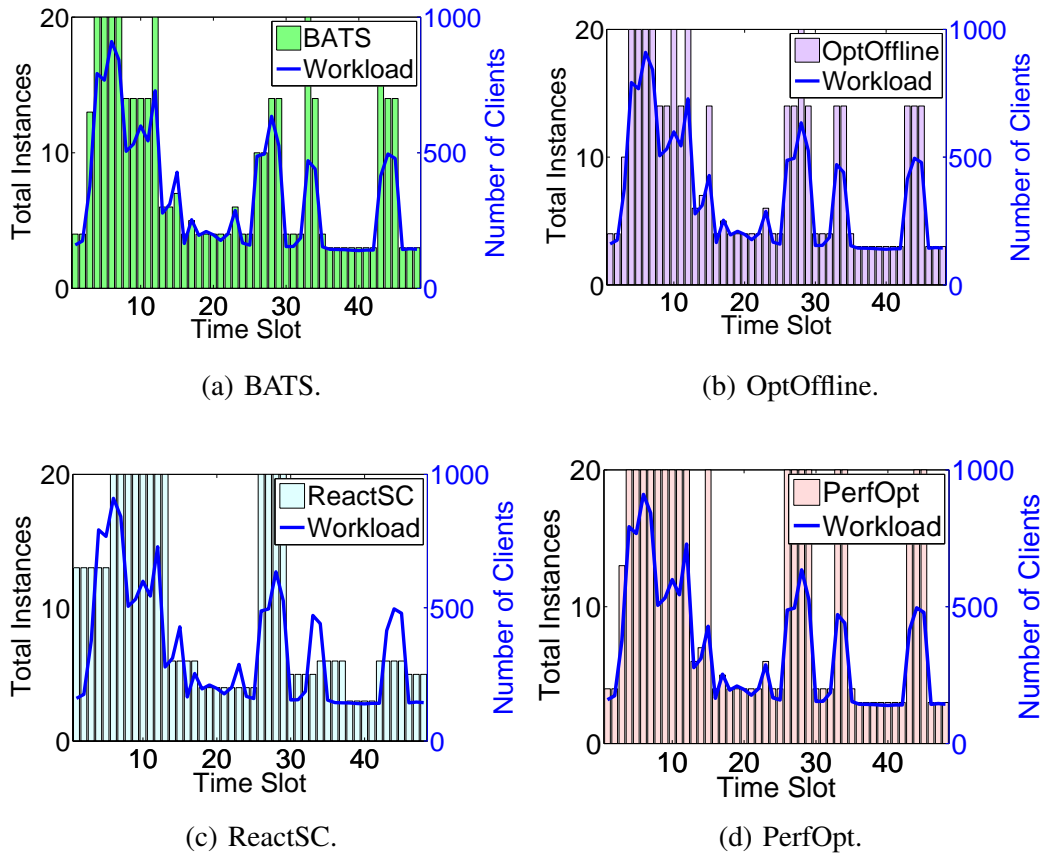


Figure 5.5: Resource allocation comparison.

- EqualSC:** The algorithm evenly divides the available budget across all the time slots and obtains the number of VM instances that can be reserved for the entire budgeting period based on discounted pricing (for reserved instances). We consider 20% discount as offered by Windows Azure [9].

- ReactSC:** Reactive scaling rules are widely adopted by both developers and third-party solution providers [1, 4–6, 9]. Most reactive scaling rules are defined by comparing a performance metric to a specific threshold. For example, add a new VM instance when the average CPU utilization exceeds 85% , or terminate a VM instance when the average CPU utilization falls below 45%. We implement a reactive autoscaler that can use more complex rules rather than simple threshold-based comparisons. The autoscaler constantly

monitors the average workload arrival rates measured over the last 5 minutes. Then, based on the monitored workload, ReactSC uses the delay lookup table to determine the minimum number of VMs so the resulting delay is equal to the the desired delay  $d_{min}$  for the upcoming time slot.

- **PerfOpt:** This algorithm knows (using perfect short-term prediction) the workload arrival rate at the beginning of each time slot and uses the delay lookup table to determine the minimum number of VM instances such that the resulting delay is equal to the desired delay  $d_{min}$ . It always optimizes performance while disregarding the desired budget constraint. Compared with ReactSC and BATS, PerfOpt assumes perfect short-term prediction information.

- **OptOffline:** The optimal offline algorithm has the perfect workload arrival information for the entire budgeting period at the very beginning. Based on complete offline information, the whole budget is optimally divided among time slots by solving **P31** based on Lagrangian technique and choosing (through bisection search) the optimal Lagrangian multiplier to ensure equality for budget constraint [17]. Essentially, the optimal Lagrangian multiplier corresponds to the budget deficit queue, but it is a fixed value, which can only be obtained based on complete offline workload information. OptOffline is not possible to implement in practice. It only serves as a reference of theoretical optimal.

Fig. 5.4 shows the experimental results. Fig. 5.4(a) and 5.4(b) compare the cumulative average delay and cumulative cost of BATS, respectively. The cumulative average for a time slot  $t$  is the corresponding average value of time slot 0 to  $t$ .

Firstly, we compare BATS to EqualSC. As shown in Fig. 5.4(a), BATS reduces delay by 34% compared to EqualSC while achieving the same budget constraint, even though EqualSC receives discounted pricing. This is mainly because EqualSC evenly divides the budget across each time slot and reserves VM instances without considering the workload

variation. The number of reserved instances is 11 for the whole budgeting period regardless of the workload variations. As a result, when there is a workload spike (e.g., in the 4<sup>th</sup> time slot), the delay becomes very large.

Secondly, we compare BATS with the widely-adopted ReactSC autoscaling mechanism. Fig. 5.4(a) shows that the average delay reduction of BATS is 10% compared to ReactSC. The degrading performance of ReactSC comes from the long *lagging* time: it takes up to 5 minutes to detect the system status change (e.g., workload variation) and even after detection, it takes up to 10 minutes to acquire a new VM instance. During the lagging time, all the incoming workloads experience longer average delays. For example, ReactSC experiences higher delay in the 25<sup>th</sup> and 42<sup>nd</sup> time slots because of its inability to cope instantaneous traffic spikes, as shown in Fig. 5.5(c). This demonstrates the importance of proactively predicting the near-future (e.g., hour-ahead) workloads as used by BATS, thereby highlighting the limitations of reactive autoscaling rules. When using ReactSC, keeping an additional resource margin/headroom (i.e., requesting more VM instances than needed) may mitigate excessive temporal delays, but will result in an even higher cost and more likely violate the budget constraint. Moreover, Fig. 5.4(b) shows that the cost saving of BATS is 10% compared to ReactSC. It is mainly because ReactSC ignores the budget constraint and always makes scaling decisions such that resulting average delay equals  $d_{min}$ . This shows that BATS outperforms ReactSC in terms of both delay reduction and cost savings.

Thirdly, we compare BATS with PerfOpt. Fig. 5.4(a) shows that although PerfOpt takes 4.4% lower average delay, its resulting cost is 16.8% higher than the user specified budget. This is mainly because PerfOpt only focuses on minimizing the delay without considering budget constraint. The achieved delays of BATS and PerfOpt are 520ms and 543ms, respectively. The additional 23 ms delay does not change the human perception of the web performance, as shown in prior study [85]. Moreover, PerfOpt assumes perfect

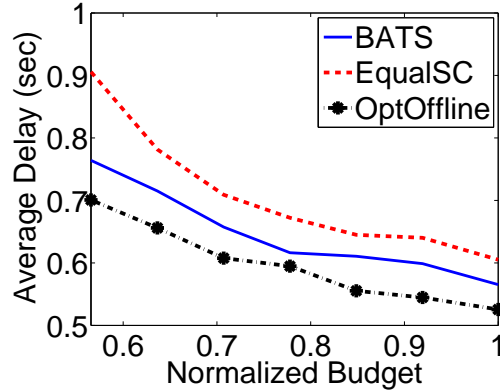


Figure 5.6: Impact of users budget.

short-term prediction information that may not be available in practice, while BATS only requires a simple predictor as described in Section 5.4.2. In summary, BATS satisfies the budget constraint while achieving a similar delay performance.

Finally, we compare BATS with the optimal offline algorithm OptOffline. Fig. 5.4(a) shows that the average delay of BATS is very close to OptOffline (with a difference less than 4%), while Fig. 5.4(b) shows that the cost is almost the same. The results demonstrate the effectiveness of BATS: it only uses the estimated workload of the next time slot, it already performs almost as well as the optimal offline autoscaler that requires the complete future prediction.

### Impact of user budget

We study how user budget affects the behavior of BATS and other benchmarks described earlier. Results show that the delay produced by BATS is never more than 10% compared to that of OptOffline for different users budget, and it is always smaller than EqualSC. We do not show the results of ReactSC and PerfOpt since they are budget-unaware and their performance is independent of the user budget.

We first describe the choice of our budget amount that will be used to benchmark the comparison between BATS and other algorithms. The highest budget value is chosen

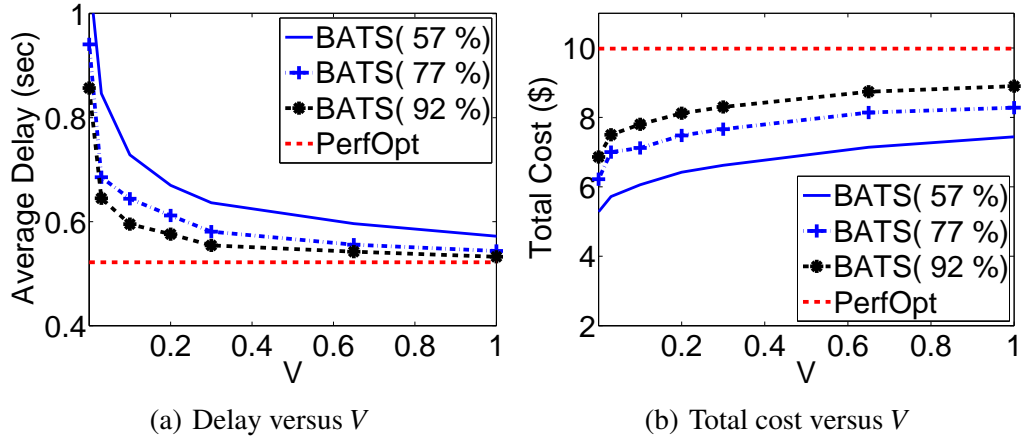


Figure 5.7: Impact of  $V$  on the delay and total cost.

based on the incurred cost of PerfOpt (i.e., always selecting the minimum number of VMs such that resulting delay for each time slot is no greater than the predetermined delay threshold  $d_{min}$ ). We normalize the actual budget by dividing it by the highest budget.

Now, we discuss the impact of user budget with three observations. (1) As shown in Fig. 5.6, the less budget, the higher delay, which matches our expectation. (2) The delay is reduced relatively more rapidly as the budget increases from 55% to 75%, and the delay reduction slows down with further increase of user budget. Under a low budget, few VM instances are used in most of the time slots, resulting in long request waiting time. The long waiting time can be effectively reduced by adding more VM instances with additional budget. However, when the budget increases further, the waiting time becomes smaller and the request execution time dominates the total delay. At this stage, the delay reduction by adding more VMs becomes smaller. (3) The delay produced by BATS is not more than 10% compared to the offline optimal for any budget constraint in the experiment, and it is always less than that of EqualSC. This shows the robust performance of BATS for all budget levels.

## Delay-cost tradeoff

This experiment discusses how the value of delay-cost parameter  $V$  affects BATS in terms of the delay-cost tradeoff under various normalized budget constraints (indicated in the parenthesis to right of “BATS” in Fig. 5.7). The result in Fig. 5.7 is consistent with our analysis: with a greater  $V$ , BATS tends to minimize the delay and becomes less concerned about the total cost. The weight of the budget deficit queue becomes less effective. If  $V$  goes to infinity, BATS becomes purely performance-driven to minimize the average delay while ignoring budget constraint, and hence reduces to budget-unaware PerfOpt. As a result, the average delay becomes  $d_{min}$ . With a smaller  $V$ , the budget deficit queue plays a more important role and BATS cares more about the cost. Fig. 5.7(b) shows that for very small  $V$ , the delay becomes very large (close to  $d_{max}$ ) and the cost is even less than the specified budget. Note that the delay of the PerfOpt algorithm represents the minimum delay that can be possibly achieved by any algorithm. As shown in Fig. 5.7, when  $V \geq 0.4$ , the average delay achieved by BATS is fairly close to  $d_{min}$ , while still satisfying the budget constraint. At this point, BATS perfectly balances between performance and budget constraint. Section 5.6.4 shows how BATS adapts  $V$  autonomously.

## 5.6 Simulation Study

This section presents simulation results of BATS, which complement the implementation results and evaluate other important aspects of an autoscaling algorithm. We first validate our simulator by comparing the simulation results with the implementation results. Secondly, we test the effectiveness and efficiency of BATS by scaling a workload requiring a few hundred VMs, and evaluating both average and tail latency. Moreover, we show that BATS (1) decides the delay-cost parameter  $V$  autonomously, (2) builds and adapts

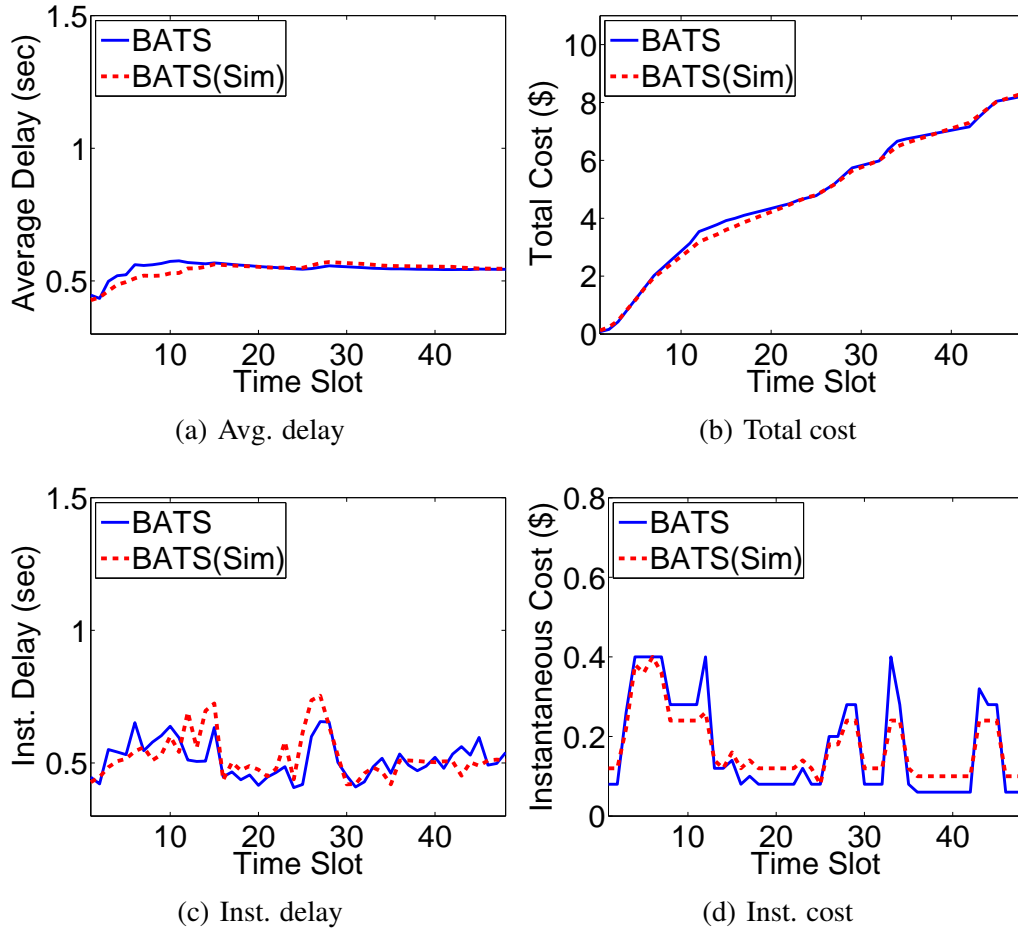


Figure 5.8: Simulator validation.

the delay lookup table online, without requiring user inputs and with negligible impact on performance and cost, and (3) is robust to prediction errors.

### 5.6.1 Simulator Overview and Validation

We develop a discrete-event based simulator based on CloudSim [20] that supports modeling and simulation of large-scale cloud computing environments. Our simulator models a set of virtual machines and arrival jobs, as well as the autoscaling algorithms that connect them.

We validate our simulator against the system experiments and show the results in Fig. 5.8. We keep all the simulation settings and parameter values the same as the system experiment for the validation purpose. Fig. 5.8(a) and Fig. 5.8(b) show that both the average delay and cost of the simulated results are very close to the experimental results (relative difference is less than 1%). Furthermore, even the instantaneous delay and cost (Fig. 5.8(c) and Fig. 5.8(d)) are quite similar. Therefore, we conclude that the simulation results closely follow the system implementation results, validating our simulator.

### 5.6.2 Experimental Setup

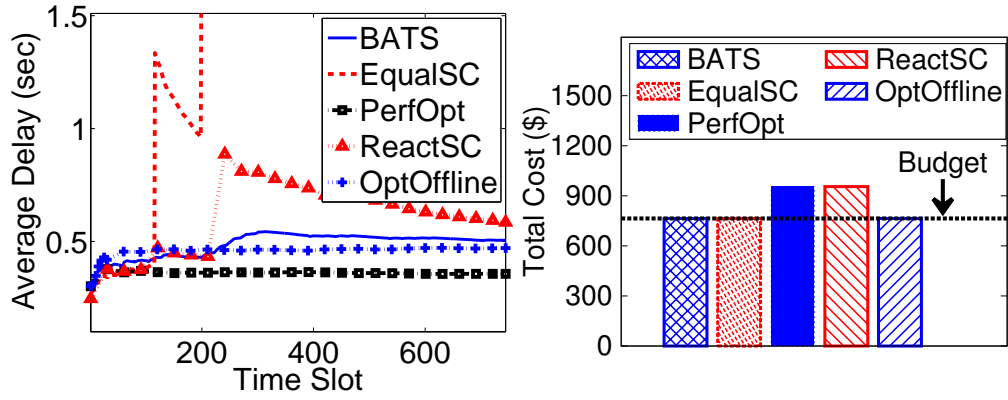
We create a virtualized data center, where each server has 6 CPU cores and 16GB of RAM. Each VM has one core and 1024MB of RAM. Our simulator has a workload generator that mimics the RUBiS workload generator. We evaluate BATS using the workload trace from Florida International University, shown in Fig. 5.2(b). We obtain this trace by profiling the web server usage logs from January 1 to January 31, 2012.

Our simulation uses the following default settings unless specified otherwise. We set the desired average delay  $d_{min} = 400$  ms, and the maximum tolerable average delay  $d_{max} = 1500$  ms. The cost per VM is \$0.02. We model a budget period of 1 month, and a total budget of \$764.

### 5.6.3 Optimizing Average and Tail Delay

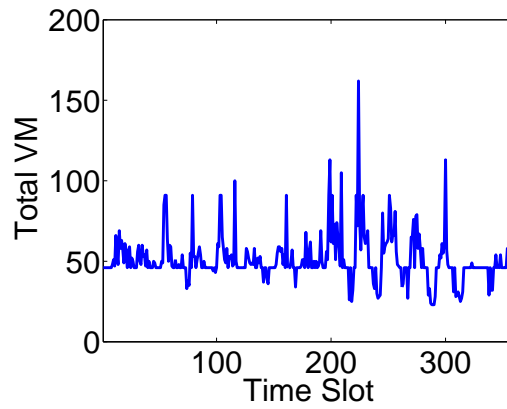
We compare BATS to the benchmark algorithms defined in Section 5.5.2. Fig. 5.9(a) and Fig. 5.9(b) compare the cumulative average delay and total cost of BATS, respectively. The results in Fig. 5.9 show that BATS outperforms other online algorithms. These results are rather consistent with the implementation results in Fig. 5.4, and hence, we skip the detailed discussion.





(a) Average delay.

(b) Total cost.



(c) Total VM instances of BATS.

Figure 5.9: Average delay comparison with other algorithms.

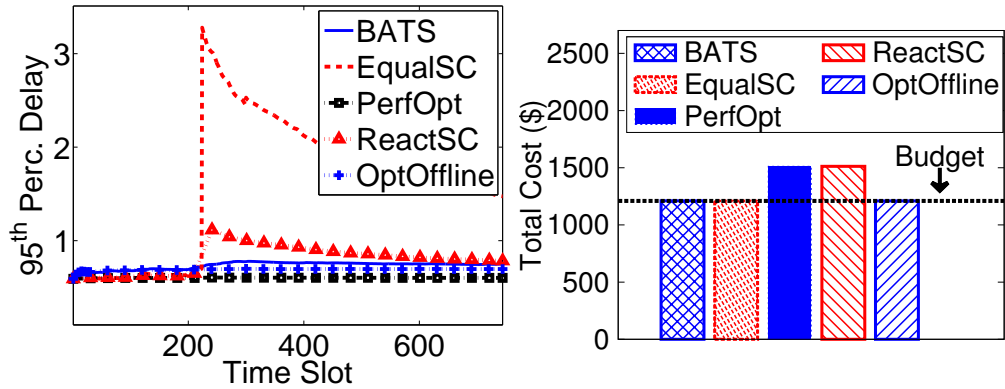


Figure 5.10: Tail delay comparison with other algorithms.

We also compare the performance of BATS in terms of 95<sup>th</sup>-percentile delay. Here we set  $d_{min} = 600$  ms and  $d_{max} = 1500$  ms, both in terms of 95<sup>th</sup>-percentile delay. Fig 5.10 shows that BATS consistently outperforms the 3 online algorithms and achieves close-to-optimal performance while satisfying budget constraint. The 95<sup>th</sup> percentile delay of BATS is 96% and 5% lower than EqualSC and ReactSC, respectively. ReactSC and PerfOpt violate the budget constraint and incur 24% more cost than the specified budget.

Moreover, the simulation is conducted to autoscale an application using more than 100 VMs (Fig. 5.9(c)). BATS takes less than 3ms to compute the allocation at each time slot. We also test BATS to solve problems with a few hundred VMs, and it takes less than 15ms, demonstrating the efficiency and scalability of BATS on solving large problems.

#### 5.6.4 Choosing Delay-Cost Parameter $V$ Autonomously

To evaluate the effectiveness of our proposed rule in Eqn. 5.28 for autonomously updating the cost-delay parameter  $V$ , we introduce a variant of BATS algorithm called BATS- $x$  which starts with an initial  $V$  value of  $x$ . The adaptation rate  $\beta$  in Eqn. 5.28 is set to 15, and  $V$  is adjusted after every 6 time slot. Fig. 5.11(a) shows that even if BATS starts with a very large or small value of  $V$ , it gradually converges and satisfies budget constraint. For example, the desired value of  $V$  for this workload is around 5, which can be measured empirically. Fig. 5.11(b) shows that when BATS-100 starts with an initial  $V$  of 100, it self-adapts and eventually becomes close the desired  $V$  value. As shown in Fig. 5.11(a), the corresponding delay till 360<sup>th</sup> time slot is less than that of BATS because of higher  $V$  values. During these time slots, the incurred cost of BATS-100 is higher than the reference budget. However, the  $V$  of BATS-100 continues to decrease until average cost per slot becomes higher than average allocated budget per slot. Thus, BATS-100 can dynamically adjust  $V$  without requiring any user input. While adapting  $V$ , the resulting delay of BATS-100 for the whole budgeting period is only 3.6% higher than

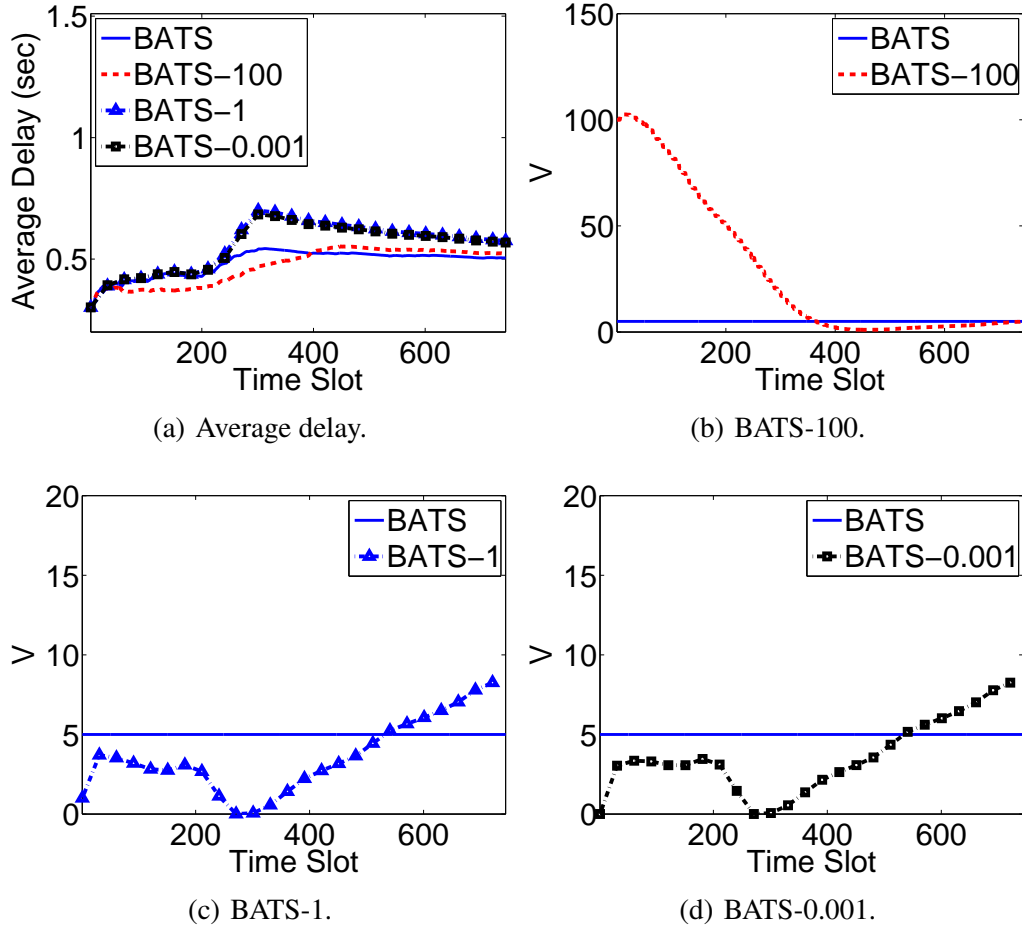


Figure 5.11: Adaptive  $V$ .

that of BATS. We also study the behavior of adaptive BATS in case a user starts with a very small value of  $V$ . Fig 5.11(a) shows that the average delay of BATS-0.001 for the whole budgeting period is 12% higher than that of BATS. These results show that BATS dynamically adapts  $V$ , robust to the setting of the initial parameters.

### 5.6.5 Learning Delay Lookup Table Online

Delay lookup table is an important input to BATS, which maps load to delay (Section 5.3.1). This table can be calibrated offline, as shown in Section 5.5.1. Now, we show how to learn the delay lookup table online.

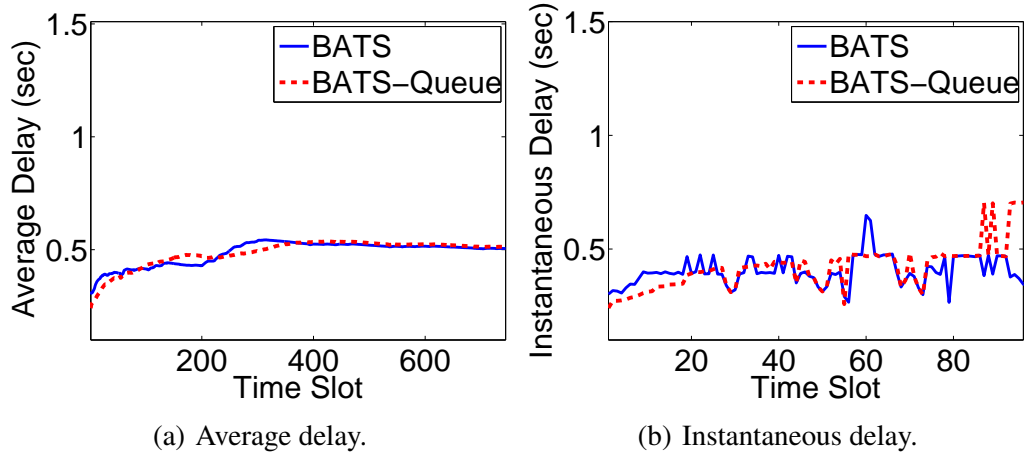


Figure 5.12: Delay performance with adaptive delay lookup table.

To populate initial values in the delay lookup table, we use queueing-theoretic models as a widely-used good *approximation* for characterizing delay performance [66, 91]. For example, we can approximate the VM service process as an M/M/1 queue, for which average delay only depends on two inputs: (1) service rate, i.e., the number of requests that can be processed by a VM in a unit time; and (2) request arrival rate. In our study, we can obtain the service rate by pre-running the cloud service on a VM for a short period of time and measuring the saturated throughput under heavy loads. Then, the delay lookup table is fulfilled where each element corresponds to a different combination of arrival rate and number of VM instances. At runtime, the table is updated continuously using the observed delay.

Fig. 5.12(a) shows that the average delay of starting from a delay lookup table initially populated with an M/M/1 queueing model (BATS-Queue) is only 1.5% higher than that of BATS while satisfying the budget constraint. By using this simple approach to update lookup table online, BATS learns the delay values autonomously and adapts to workload variations during the budget period.

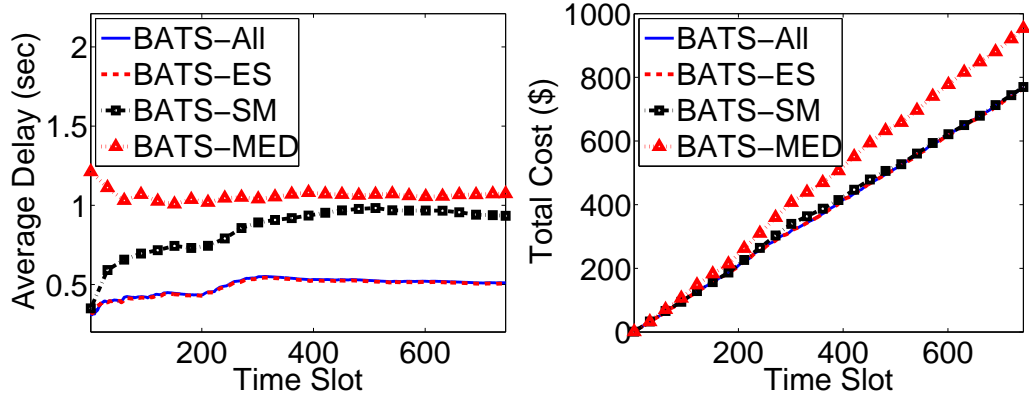


Figure 5.13: Impact of horizontal and vertical scaling.

### 5.6.6 Selecting Among Different Types of VMs.

We study how BATS performs with both vertical scaling (scaling up/down) and horizontal scaling (scaling in/out). Vertical scaling means increasing (up) or decreasing (down) the amount of resources (e.g., virtual CPU, memory and storage) allocated to a VM instance. Horizontal scaling means adding (out) or reducing (in) the number of VM instances. In practice, CSPs often offer multiple types and sizes of VMs to meet varying demand of applications. This experiment shows that, using the delay lookup tables of the application running on different types of VMs, BATS selects the proper VM instance type to minimize cost and average delay. To build the delay lookup tables of using different types of VMs, we can either calibrate the application performance offline or model the application performance online as suggested by prior work [50, 61, 109].

We compare the performance of **BATS-All** — which can use any type of VMs, supporting both vertical and horizontal scaling — with **BATS-ES**, **BATS-SM** and **BATS-MED**, which use only extra small, small and medium VM instances respectively. Fig. 5.13(a) shows that the performance of BATS-All is exactly the same as the performance of BATS-ES. This is because, for the modeled workload, the extra small VM instances have the highest cost effectiveness: the cost of small VM instance is 4 time higher than

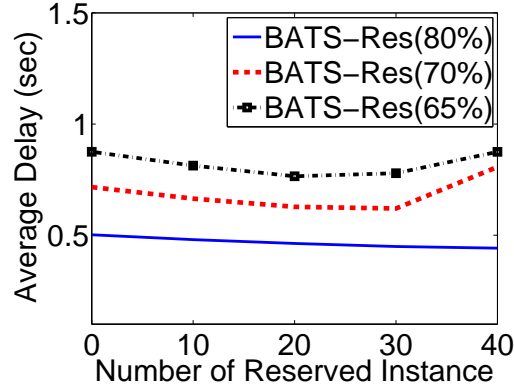


Figure 5.14: Impact of Reserved Instances on Average delay.

that of an extra small VM instance while the performance gain for RUBiS workload is only 3 times. Similarly, for medium VM instances, cost is 8 times higher while the performance gain is 5 times. Because of the cost effectiveness of the extra small VM instances, BATS-All always chooses it. The cost reduction of BATS-All is 82% and 100% compared with the BATS-SM and BATS-MED, respectively. Thus by using the delay lookup tables, BATS chooses the right type and number of the VM instances for the application, supporting both vertical and horizontal scaling.

### 5.6.7 Supporting Reserved Instances.

We explore how reserve instances can be used to improve delay performance while satisfying the budget constraint. We discuss how to incorporate reserved instances into our model in section 5.3.5. Reserved instances are cheaper than on-demand instances. For example, Azure provides 20% to 27% discounts depending on the purchase volume for 6 month purchase plans. Thus, users can leverage the lower price of reserve instances and save costs while achieving delay reduction. We introduce a variation of BATS algorithm **BATS-Res** which reserves *Extra Small* VM instances at the beginning of the budgeting period and uses rest of the budget for autoscaling. We consider 3 budget levels for **BATS-Res**: 80%, 70% and 65% of the reference budget.

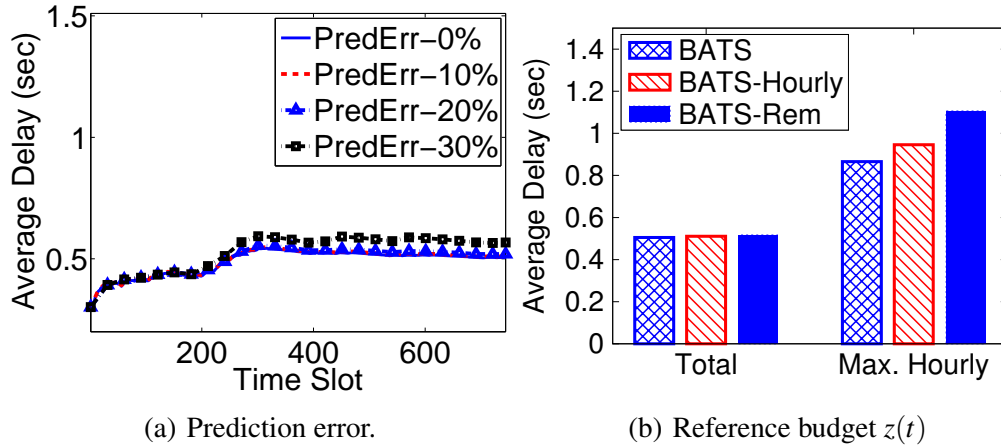


Figure 5.15: Sensitivity study.

Fig. 5.14 shows how the number of reserved instances affects average delay under different budget levels. The main observation here is for any budget level, the average delay is minimized if certain amount of VM instances are reserved. For example, when the budget level is 70% of the reference budget, the maximum delay reduction is 13.5% when 30 instances are reserved compared with that of no reserved instances. This reason behind is intuitive. An appropriate number of reserved instances can save costs while delivering the same computing power as on-demand instances. The saved costs can be used by autoscaler when the workload is high. The challenge is to decide the amount of reservation such that it becomes cost effective in the long run. If too many instances are reserved, they may be under-utilized at the lightly-loaded interval of the budgeting period. In such case, despite being cheaper, they are wasted. Determining an optimal number of reservation requires long-term prediction information and studied in [16, 47, 108], which is beyond the scope of our work. However, once the reserved instances and the remaining budget are decided, BATS can incorporate these information and autoscale the dynamic instances to meet the remaining budget and minimize average delay.

### 5.6.8 Sensitivity Study

**Prediction errors:** As BATS leverages the hour-ahead workload prediction, we evaluate how BATS performs in the presence of prediction errors. We consider four cases where the workload predictor introduces prediction errors of  $0\%$ ,  $\pm 10\%$ ,  $\pm 20\%$  and  $\pm 30\%$ . Fig.5.15(a) shows that, compared to  $0\%$  prediction error, the resulting delay increases by  $0.9\%$ ,  $2.8\%$  and  $12.3\%$  for  $\pm 10\%$ ,  $\pm 20\%$  and  $\pm 30\%$  prediction errors, respectively. Even with high prediction errors, the performance of BATS is still quite robust. Moreover, in practice, the reactive part of BATS also compensates for the prediction error and ensure that the delay does not violate user requirements. Thus, BATS can be successfully applied even when the workload prediction is not perfect.

**Reference budget  $z(t)$ :** We explore the impact of different choices of reference budget  $z(t)$  in Eqn. 5.8. In our above studies, the reference budget  $z(t)$  at time slot  $t$  in BATS is obtained by dividing the whole budget evenly to each time slot. For comparison purposes, we consider two variants of BATS, where we set reference budget  $z(t)$  by: (1)**BATS-Rem:** evenly dividing the remaining budget at time slot  $t$ ; (2) **BATS-Hourly:** dividing the budget to each time slot according to the average workload arrival rate obtained from past data. Fig. 5.15(b) shows that while choosing  $z(t)$  differently, the delay performance remains relatively the same with less than  $2\%$  difference. Intuitively, the reference budget  $z(t)$  in Eqn. 5.8 only directly impacts the runtime budget deficit queue dynamics, thereby not being enforced as runtime budget constraint or directly impacting the autoscaling decision. Thus, in the long-term, as long as the total budget is the same,  $z(t)$  has a negligible impact on the delay performance, demonstrating the robustness of BATS against the choice of  $z(t)$ .



## 5.7 Summary

This chapter provides a full-fledged autoscaling solution, BATS, to optimize delay performance while meeting users' long-term budget constraints using only past and instantaneous workload information. Analytically, we proved that the autoscaling algorithm of BATS achieves a close-to-optimal performance even compared to the optimal solution that has complete offline information. We implemented BATS autoscaler as an automated service for cloud applications on Windows Azure, and we conducted extensive experimental and simulation studies. The empirical results further demonstrate the effectiveness, efficiency, autonomy, and robustness of BATS on a wide range of scenarios with various workloads and scaling capabilities.

## CHAPTER 6

### CONCLUSION

In this chapter, we provide a brief summary of the dissertation and present some possible direction for future work.

#### 6.1 Summary of the Dissertation

In the view of increasing data center carbon emission and energy consumption, this dissertation proposes some novel approaches to improve sustainability of data center operation. In the first part of the dissertation, we present how to cap and reduce the carbon emission of a self-managed data center to “net-zero” level. We design an online resource management algorithm, CNDC, to minimize the operational cost of a data center with demand-responsive electricity prices while achieving carbon neutrality for data centers, in the presence of time-varying workloads and intermittent renewable energy supplies. We extend the existing Lyapunov optimization technique by enabling dynamic adjustment of the control parameter (which governs the tradeoff between cost minimization and the potential deviation from carbon neutrality in our study) while still being able to offer an analytical performance bound. We perform both trace-based simulation and system experimentation to evaluate CNDC and demonstrate its effectiveness

In the next part, we extend our problem of carbon emission reduction for a hybrid data center infrastructure that includes both self-managed and colocation data centers. We identify the problem and challenges of minimizing the carbon emission in a hybrid data center infrastructure (which are very common in practice for supporting large organizations’ computing needs). We propose a novel and distributed geographical job scheduling algorithm to reduce carbon footprint and electricity cost of hybrid data centers. Our ex-

tensive simulation studies and system experiments show the effectiveness of our proposed solution.

Finally, we discuss how to address sustainability from a cloud service user's side. We build a *fully-automated* BATS autoscaler as a user-friendly service for running applications on Windows Azure. BATS takes care of the performance monitoring, resource planning, self-adjustment, and scaling automatically; it only needs user inputs on their desired delay performance and budget, along with some basic application information. We formally prove that the BATS autoscaling algorithm produces a close-to-optimal delay performance compared to the optimal algorithm with offline information. We evaluate the performance of BATS by running RUBiS benchmark workloads [89] in Windows Azure Cloud Service, as well as using extensive simulations.

Many organizations (e.g., Apple, Facebook and Google) are continuously looking forward to reduce their energy consumption and carbon emission to mitigate the pressure from utility companies, qualify for government incentives, and improve their public image. Our work has a potential to improve carbon emission by thousands of tons, reduce energy consumption, and save millions of dollars by cutting the electricity cost of an IT organization.

## **6.2 Future Direction**

In the first part of the dissertation, we discuss how to reduce carbon emission and achieve carbon neutrality for a self-managed data center using Lyapunov optimization technique. Achieving carbon neutrality in geographically distributed data centers can be a challenging future work. It will be interesting to see how Lyapunov optimization works in a distributed environment. Furthermore, our model in this part focuses on a single-tier web application, and it will be a challenging future direction to extend it for a multi-tier web application.

In the second part of the dissertation, we present how to reduce the carbon emission of a hybrid data center infrastructure. Resource management and workload scheduling in a hybrid data center infrastructure is largely an unexplored field. There are many opportunities to extend the work presented in this dissertation from an organization's perspective.

1. Optimizing the organization's bid collectively, in multiple colocation data centers when participating in emergency demand-response program, is an interesting future work. Note that, in emergency demand response (EDR) program, the utility provider offers financial incentives to the businesses that reduce their electricity consumption upon receiving a signal.
2. Achieving carbon neutrality in hybrid infrastructure is also a potential direction. It involves modification and extension of Lyapunov optimization technique such for a distributed environment.
3. Integrating emergency demand response program and temperature aware workload placement in a hybrid data center environment is another possible direction. In self-managed data centers, the organization can turn on idle servers and adjust the cooling temperature to reduce energy consumption. On the other hand, the operator manages the cooling system in the colocation facility and have no control over the tenants server. The colocation operator may increase cooling temperature to reduce energy consumption, but it will cause overheating of other tenant's servers. Thus, traditional approaches for temperature-aware workload placement can not be applied to a hybrid data center infrastructure, mainly because of the colocation part.
4. Our study focuses on delay-sensitive interactive workloads. A possible extension is to consider delay-tolerant batch workloads. Unlike interactive jobs, batch jobs can be deferred as long as they complete before the deadline. This presents a unique

opportunity to reduce operational costs, carbon emission, and energy consumption by delaying a batch job until a time slot which has lower electricity price, higher carbon efficiency, and higher cooling efficiency, respectively. However, deferring the batch job processing makes the resource management decision quite challenging as it couples multiple time slots together. Without knowing or predicting the future information accurately, delaying batch jobs from the current slot to next may overload subsequent slots.

In the final part of the dissertation, we address economical and environmental sustainability issues from a cloud service user's perspective. Our study focuses on a single-tier web application. An exciting future work is to extend it for a multi-tier application which presents significantly higher challenges for modeling delay and performance. Furthermore, reducing carbon emission directly by predicting the CSP's VM placement and corresponding carbon emission can be another direction.

## BIBLIOGRAPHY

- [1] Amazon ec2. <http://aws.amazon.com/autoscaling/>.
- [2] Azure cloud service. <http://www.windowsazure.com/en-us/manage/services/cloud-services/what-is-a-cloud-service/>.
- [3] California ISO, <http://www.caiso.com/>.
- [4] Rackspace. <http://www.rackspace.com/>.
- [5] Rightscale. <http://www.rightscale.com/>.
- [6] Scarl. <http://www.scalr.com/>.
- [7] What epa is doing about climate change. <https://www3.epa.gov/climatechange/EPAactivities.html>.
- [8] Where carbon is taxed. <http://www.carbontax.org/where-carbon-is-taxed/>.
- [9] Windows azure. <http://www.windowsazure.com/>.
- [10] Amazon ec2 spot instances, <http://aws.amazon.com/ec2/spot-instances/>.
- [11] Hrishikesh Amur, Ripal Nathuji, Mrinmoy Ghosh, Karsten Schwan, and Hsien-hsin S Lee. Idlepower: Application-aware management of processor idle states. In *In Proceedings of MMCS, in conjunction with HPDC'08*, 2008.
- [12] Apple. Environmental responsibility report, 2014, [http://images.apple.com/environment/reports/docs/Apple\\_Environmental\\_Responsibility\\_Report\\_2014.pdf](http://images.apple.com/environment/reports/docs/Apple_Environmental_Responsibility_Report_2014.pdf).
- [13] AT&T. Network latency. [http://ipnetwork.bgtmo.ip.att.net/pws/network\\_delay.html](http://ipnetwork.bgtmo.ip.att.net/pws/network_delay.html).
- [14] Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, (12):33–37, 2007.
- [15] W Lloyd Bircher and Lizy K John. Complete system power estimation using processor performance events. *IEEE Transactions on Computers*, 61(4):563–577, 2012.

- [16] Christian Bodenstein, Markus Hedwig, and Dirk Neumann. Strategic decision support for smart-leasing infrastructure-as-a-service. *ICIS*, 2011.
- [17] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.
- [18] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.
- [19] Ulrich Brenner. A faster polynomial algorithm for the unbalanced hitchcock transportation problem. *Operations Research Letters*, 36(4):408–413, 2008.
- [20] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.
- [21] Gong Chen, Wenbo He, Jie Liu, Suman Nath, Leonidas Rigas, Lin Xiao, and Feng Zhao. Energy-aware server provisioning and load dispatching for connection-intensive internet services. In *NSDI*, 2008.
- [22] Data Center Knowledge. Colos spend more on data centers as enterprises tighten purse strings. <http://www.datacenterknowledge.com/>.
- [23] Nan Deng, Christopher Stewart, Daniel Gmach, Martin Arlitt, and Jaimie Kelley. Adaptive green hosting. In *ICAC*, 2012.
- [24] Nan Deng, Christopher Stewart, Daniel Gmach, and Martin F. Arlitt. Policy and mechanism for carbon-aware cloud applications. In *NOMS*, 2012.
- [25] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F Wenisch, and Ricardo Bianchini. Memscale: active low-power modes for main memory. *ACM SIGARCH Computer Architecture News*, 39(1):225–238, 2011.
- [26] Yixin Diao, Joseph L Hellerstein, Sujay Parekh, Rean Griffith, Gail E Kaiser, and Dan Phung. A control theory foundation for self-managing computing systems. *JSAC*, 23:2213–2222, 2005.
- [27] Tamara DiCaprio. Becoming carbon neutral: How Microsoft is striving to become leaner, greener, and more accountable. *Microsoft Whitepaper*, June 2012.

- [28] Enaxis Consulting. Pricing data center co-location services, 2009, [http://enaxisconsulting.com/downloads/2/67f7fb873eaf29526a11a9b7ac33bfac/1317636458\\_data\\_center\\_pricing.pdf](http://enaxisconsulting.com/downloads/2/67f7fb873eaf29526a11a9b7ac33bfac/1317636458_data_center_pricing.pdf).
- [29] ENERGY.GOV. Demand response. <http://energy.gov/oe/technology-development/smart-grid/demand-response>.
- [30] Archana Ganapathi, Yanpei Chen, Armando Fox, Randy Katz, and David Patterson. Statistics-driven workload modeling for the cloud. In *ICDEW*, 2010.
- [31] A. Gandhi, Yuan Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *IGCC*, 2011.
- [32] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, model-driven autoscaling for cloud applications. In *ICAC 14*, 2014.
- [33] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In *SIGMETRICS*, 2009.
- [34] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A. Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Trans. Comput. Syst.*, 30(4):14:1–14:26, November 2012.
- [35] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012.
- [36] Peter Xiang Gao, Andrew R. Curtis, Bernard Wong, and Srinivasan Keshav. It’s not easy being green. *SIGCOMM Comp. Comm. Rev.*, 42(4):211–222, August 2012.
- [37] Saurabh Kumar Garg, Chee Shin Yeo, Arun Anandasivam, and Rajkumar Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749, 2011.
- [38] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Guillaume Belrose, Tom Turicchi, and Alfons Kemper. An integrated approach to resource pool management: Policies, efficiency and quality metrics. In *DSN*, 2008.



- [39] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Capacity management and demand prediction for next generation data centers. In *ICWS*, 2007.
- [40] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *CNSM*, 2010.
- [41] Google. Google’s green PPAs: What, how, and why, [http://static.googleusercontent.com/external\\_content/untrusted\\_dlcp/www.google.com/en/us/green/pdfs/renewable-energy.pdf](http://static.googleusercontent.com/external_content/untrusted_dlcp/www.google.com/en/us/green/pdfs/renewable-energy.pdf).
- [42] Sriram Govindan, Di Wang, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. In *ASPLOS*, 2012.
- [43] Greenpeace. Clicking clean: How companies are creating the green internet. <http://www.greenpeace.org/usa/global/usa/planet3/pdfs/clickingclean.pdf>.
- [44] Greenpeace. How dirty is your data? a look at the energy choices that power cloud computing, 2011.
- [45] Brian Guenter, Navendu Jain, and Charles Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In *INFOCOM*, 2011.
- [46] Y. Guo, Z. Ding, Y. Fang, and D. Wu. Cutting down electricity cost in internet data centers by using energy storage. In *Globecom*, 2011.
- [47] Yu-Ju Hong, Jiachen Xue, and Mithuna Thottethodi. Dynamic server provisioning to minimize cost in an iaas cloud. In *ACM SIGMETRICS*, 2011.
- [48] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 1. IEEE Computer Society, 2005.
- [49] International Energy Agency. Global energy-related emissions of carbon dioxide stalled in 2014. <http://www.iea.org>.
- [50] Alexandru Iosup, Simon Ostermann, M Nezhil Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick HJ Epema. Performance analysis of cloud computing services for many-tasks scientific computing. *TPDS*, 22(6):931–945, 2011.

- [51] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.
- [52] M. A. Islam, S. Ren, N. Pissinou, H. Mahmud, and A. Vasilakos. Distributed resource management in data centers with temperature constraint. In *IGCC*, 2013.
- [53] Mohammad Islam, Shaolei Ren, A. Hasan Mahmud, and Gang Quan. Online energy budgeting for cost minimization in virtualized data center. *IEEE Transactions on Services Computing*, PP(99):1–1, 2015.
- [54] Mohammad Islam, Shaolei Ren, Xiaorui Wang, et al. Greencolo: A novel incentive mechanism for minimizing carbon footprint in colocation data center. In *IGCC*, 2014.
- [55] Mohammad A. Islam, Hasan Mahmud, Shaolei Ren, and Xiaorui Wang. Paying to save: Reducing cost of colocation data center via rewards. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.
- [56] Mohammad A Islam and Shaolei Ren. A new perspective on energy accounting in multi-tenant data centers. In *CoolDC 16*, 2016.
- [57] Mohammad A Islam, Shaolei Ren, Niki Pissinou, A. Hasan Mahmud, and Athanasios V Vasilakos. Distributed temperature-aware resource management in virtualized data center. *Sustainable Computing: Informatics and Systems*, 2014.
- [58] Mohammad A Islam, Xiaoqi Ren, Shaolei Ren, Adam Wierman, and Xiaorui Wang. A market approach for handling power emergencies in multi-tenant data center. In *HPCA*, 2016.
- [59] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *IEEE 14th International Symposium on High Performance Computer Architecture, HPCA.*, pages 123–134. IEEE, 2008.
- [60] Data Center Knowledge. Special report: The world’s largest data centers. <http://www.datacenterknowledge.com/special-report-the-worlds-largest-data-centers/>.
- [61] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application performance modeling in a virtualized environment. In *HPCA*, 2010.

- [62] Latisys. Is 2015 finally the year of the cloud? <http://www.latisys.com/>.
- [63] Brian Lavallo. Undertaking the challenge to reduce the data center carbon footprint. <http://www.datacenterknowledge.com>.
- [64] Kien Le, Ricardo Bianchini, Thu D. Nguyen, Ozlem Bilgir, and Margaret Martonosi. Capping the brown energy consumption of internet services at low cost. In *IGCC*, 2010.
- [65] Kien Le, Ricardo Bianchini, Jingru Zhang, Yogesh Jaluria, Jiandong Meng, and Thu D. Nguyen. Reducing electricity cost through virtual machine placement in high performance computing clouds. *SuperComputing*, 2011.
- [66] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, 2011.
- [67] Zhenhua Liu, Yuan Chen, Cullen Bash, Adam Wierman, Daniel Gmach, Zhikui Wang, Manish Marwah, and Chris Hyser. Renewable and cooling aware workload management for sustainable data centers. In *SIGMETRICS PER*, volume 40, pages 175–186, 2012.
- [68] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H. Low, and Lachlan L.H. Andrew. Greening geographical load balancing. In *SIGMETRICS*, 2011.
- [69] Zhenhua Liu, Adam Wierman, Yuan Chen, Benjamin Razon, and Niangjun Chen. Data center demand response: avoiding the coincident peak via workload shifting and local generation. In *SIGMETRICS*, 2013.
- [70] Xingjian Lu, Fanxin Kong, Jianwei Yin, Xue Liu, Huiqun Yu, and Guisheng Fan. Geographical job scheduling in data centers with heterogeneous demands and servers. In *CLOUD*. IEEE, 2015.
- [71] A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. Bats: budget-constrained autoscaling for cloud performance optimization. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 563–564, 2014.
- [72] A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. BATS: budget-constrained autoscaling for cloud performance optimization. In *IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2015.

- [73] A. Hasan Mahmud and S. S. Iyengar. A distributed framework for carbon and cost aware geographical job scheduling in a hybrid data center infrastructure. In *The 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016.
- [74] A. Hasan Mahmud and Shaolei Ren. Online capacity provisioning for carbon-neutral data center with demand-responsive electricity prices. *ACM SIGMETRICS Performance Evaluation Review*, 41(2):26–37, 2013.
- [75] A. Hasan Mahmud and Shaolei Ren. Online capacity provisioning for carbon-neutral data centers with demand-responsive electricity prices. In *IFIP Performance*, 2013.
- [76] A. S. M. Hasan Mahmud and S. Ren. Dynamic server provisioning for carbon-neutral data centers. In *42nd International Conference on Parallel Processing (ICPP)*, 2013.
- [77] A. S. M. Hasan Mahmud and Shaolei Ren. Online resource management for data center with energy capping. In *Feedback Computing*, 2013.
- [78] Maciej Malawski, Gideon Juve, Ewa Deelman, and Jarek Nabrzyski. Cost-and deadline-constrained provisioning for scientific workflow ensembles in iaas clouds. In *SC*, 2012.
- [79] Ming Mao and M. Humphrey. Scaling and scheduling to maximize application performance within budget constraints in cloud workflows. In *IPDPS*, 2013.
- [80] Ming Mao, Jie Li, and M. Humphrey. Cloud auto-scaling with deadline and budget constraints. In *GRID*, 2010.
- [81] Markets and Markets. Global colocation market: Worldwide market forecasts and analysis (2013 - 2018). <http://www.marketsandmarkets.com>.
- [82] Stephen McCluer and Jean-Francois Christin. Comparing data center batteries, flywheels, and ultracapacitors. *APC Whitepaper*, 2008.
- [83] Nathan D. Mickulicz, Priya Narasimhan, and Rajeev Gandhi. To auto scale or not to auto scale. In *ICAC*, 2013.
- [84] H. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia. Autonomous demand side management based on game-theoretic energy consump-

- tion scheduling for the future smart grid. *IEEE Trans. Smart Grid*, 1(3):320–331, December 2010.
- [85] Fiona Fui-Hoon Nah. A study on tolerable waiting time: how long are web users willing to wait? *Behaviour & Information Technology*, 23(3):153–163, 2004.
- [86] National Renewable Energy Laboratory. National solar radiation data base. [http://rredc.nrel.gov/solar/old\\_data/nsrdb/1991-2010/hourly/list\\_by\\_state.html](http://rredc.nrel.gov/solar/old_data/nsrdb/1991-2010/hourly/list_by_state.html).
- [87] M. J. Neely. *Stochastic Network Optimization with Application to Communication and Queueing Systems*. Morgan & Claypool, 2010.
- [88] NRDC. Scaling up energy efficiency across the data center industry: Evaluating key drivers and barriers. *Issue Paper*, 2014.
- [89] OW2 Consortium. RUBiS: Rice University Bidding System. <http://rubis.ow2.org/>.
- [90] Darshan S. Palasamudram, Ramesh K. Sitaraman, Bhuvan Uргаonkar, and Rahul Uргаonkar. Using batteries to reduce the power costs of internet-scale distributed networks. In *SoCC*, 2012.
- [91] N. U. Prabhu. *Foundations of Queueing Theory*. Kluwer Academic Publishers, 1997.
- [92] Asfandyar Qureshi, Rick Weber, Hari Balakrishnan, John Guttag, and Bruce Maggs. Cutting the electric bill for internet-scale systems. In *SIGCOMM*, volume 39, pages 123–134, 2009.
- [93] L. Rao, X. Liu, L. Xie, and Wenyu Liu. Reducing electricity cost: Optimization of distributed internet data centers in a multi-electricity-market environment. In *INFOCOM*, 2010.
- [94] Chuangang Ren, Di Wang, Bhuvan Uргаonkar, and Anand Sivasubramaniam. Carbon-aware energy capacity planning for datacenters. In *MASCOTS*, 2012.
- [95] S. Ren, Y. He, S. Elnikety, and K. S. McKinley. Exploiting processor heterogeneity in interactive services. In *ICAC*, 2013.

- [96] S Ren and MA Islam. Colocation demand response: Why do i turn off my servers. In *ICAC*, 2014.
- [97] Shaolei Ren, Yuxiong He, and Fei Xu. Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *ICDCS*, 2012.
- [98] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *CLOUD*, 2011.
- [99] Rizos Sakellariou, Henan Zhao, Eleni Tsiakkouri, and Marios D Dikaiakos. Scheduling workflows with budget constraints. In *Integrated Research in GRID Computing*, pages 189–202. 2007.
- [100] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *SoCC*, 2011.
- [101] Piyush Shivam, Shivnath Babu, and Jeffrey S Chase. Learning application models for utility resource planning. In *ICAC*, 2006.
- [102] Qihang Sun, Chuan Wu, Shaolei Ren, and Zongpeng Li. Fair rewarding in colocation data centers: Truthful mechanism for emergency demand response. In *International Symposium on Quality of Service*, 2015.
- [103] Uptime Institute. Data center industry survey, 2013, <http://uptimeinstitute.com/2013-survey-results>.
- [104] R. Urgaonkar, B. Urgaonkar, M. J. Neely, and A. Sivasubramaniam. Optimal power cost management using stored energy in data centers. In *SIGMETRICS*, 2011.
- [105] Steven VanRoekel. The FY14 President’s IT budget: Innovate, deliver, protect. <https://cio.gov/the-fy14-presidents-it-budget-innovate-deliver-protect/>.
- [106] Di Wang, Chuangang Ren, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Hosam Fathy. Energy storage in datacenters: what, where, and how much? In *SIGMETRICS*, 2012.
- [107] Peijian Wang, Lei Rao, Xue Liu, and Yong Qi. D-pro: Dynamic data center operations with demand-responsive electricity prices in smart grid. *IEEE Transactions on Smart Grid*, 3(4):1743–1754, December 2012.

- [108] Wei Wang, Baochun Li, , and Ben Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In *ICAC*, 2013.
- [109] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Middleware*, 2008.
- [110] Dan Xu, Xin Liu, and Bin Fan. Efficient server provisioning and offloading policies for internet data centers with dynamic load-demand. *Computers, IEEE Transactions on*, 64(3):682–697, 2015.
- [111] Hong Xu and Baochun Li. Reducing electricity demand charge for data centers with partial execution. In *Proceedings of the 5th international conference on Future energy systems*, 2014.
- [112] Yuan Yao, Longbo Huang, Abhihshek Sharma, Leana Golubchik, and Michael Neely. Data centers power reduction: A two time scale approach for delay tolerant workloads. In *INFOCOM*, 2012.
- [113] Jia Yu and Rajkumar Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14:217–230, 2006.
- [114] Liang Yu, Tao Jiang, and Yang Cao. Energy cost minimization for distributed internet data centers in smart microgrids considering power outages. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):120–130, 2015.
- [115] Qi Zhang, Ludmila Cherkasova, Guy Mathews, Wayne Greene, and Evgenia Smirni. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Middleware*. 2007.
- [116] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware*, 2011.
- [117] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Electricity bill capping for cloud-scale data centers that impact the power markets. In *ICPP*, 2012.
- [118] Jian Zhao, Hongxing Li, Chuan Wu, Zongpeng Li, Zhizhong Zhang, and Francis Lau. Dynamic pricing and profit maximization for the cloud with geo-distributed data centers. In *INFOCOM*. IEEE, 2014.

- [119] Wenli Zheng, Kai Ma, and Xiaorui Wang. Exploiting thermal energy storage to reduce data center capital and operating expenses. In *HPCA*, 2014.
- [120] Z. Zhi, F. Liu, Y. Xu, R. Zou, H. Xu, J. C. S. Lui, and H. Jin. Carbon-aware load balancing for geo-distributed cloud services. In *MASCOTS*, 2013.
- [121] Zhi Zhou, Fangming Liu, Zongpeng Li, and Hai Jin. When smart grid meets geo-distributed cloud: An auction approach to datacenter demand response. In *INFOCOM*, 2015.
- [122] Jun Zhu, Zhefu Jiang, and Zhen Xiao. Twinkle: A fast resource provisioning mechanism for internet services. In *INFOCOM*, 2011.



## VITA

A. S. M. HASAN MAHMUD

January 1, 1986	Born, Chandpur, Bangladesh
2003–2007	B.S., Computer Science and Engineering Bangladesh University of Engineering and Technology Dhaka, Bangladesh
2011–2014	M.S., Computer Science Florida International University Miami, Florida
2011–2016	Doctoral Candidate Florida International University Miami, Florida

## PUBLICATIONS

A. Hasan Mahmud and S. S. Iyengar. A distributed framework for carbon and cost aware geographical job scheduling in a hybrid data center infrastructure. In *The 13th IEEE International Conference on Autonomic Computing (ICAC)*, 2016.

A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. BATS: budget-constrained autoscaling for cloud performance optimization. In *IEEE 23rd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2015.

Mohammad Islam, Shaolei Ren, A. Hasan Mahmud, and Gang Quan. Online energy budgeting for cost minimization in virtualized data center. *IEEE Transactions on Services Computing*, PP(99):1–1, 2015.

Mohammad A. Islam, Hasan Mahmud, Shaolei Ren, and Xiaorui Wang. Paying to save: Reducing cost of colocation data center via rewards. In *IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

A. Hasan Mahmud, Yuxiong He, and Shaolei Ren. Bats: budget-constrained autoscaling for cloud performance optimization. In *ACM SIGMETRICS Performance Evaluation Review*, volume 42, pages 563–564, 2014. (Poster).

Mohammad A Islam, Shaolei Ren, Niki Pissinou, A. Hasan Mahmud, and Athanasios V Vasilakos. Distributed temperature-aware resource management in virtualized data center. *Sustainable Computing: Informatics and Systems*, 2014.

A. Hasan Mahmud and Shaolei Ren. Online capacity provisioning for carbon-neutral data centers with demand-responsive electricity prices. In *IFIP Performance*, 2013.

A. Hasan Mahmud and Shaolei Ren. Online capacity provisioning for carbon-neutral data center with demand-responsive electricity prices. *ACM SIGMETRICS Performance Evaluation Review*, 41(2):26–37, 2013.

A. S. M. Hasan Mahmud and S. Ren. Dynamic server provisioning for carbon-neutral data centers. In *42nd International Conference on Parallel Processing (ICPP)*, 2013.

A. S. M. Hasan Mahmud and Shaolei Ren. Online resource management for data center with energy capping. In *Feedback Computing*, 2013. (Best Paper Award).

M. A. Islam, S. Ren, N. Pissinou, H. Mahmud, and A. Vasilakos. Distributed resource management in data centers with temperature constraint. In *IGCC*, 2013.