Energy Cost Optimization in Large Scale Distributed Systems by Resource Allocation Techniques

by

Hüseyin Güler

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

Koç University

September 9, 2013

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Hüseyin Güler

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. Öznur Özkasap (Advisor)

_____

Assoc. Prof. Yücel Yemez

_____

Prof. A. Murat Tekalp

Date: _____

To my family.

ABSTRACT

Large scale distributed systems require massive amount of computing power to provide reliable services and to solve computationally complex problems. In that regard, energy needs in these systems are increasing rapidly and that brings substantial costs. Thus, reducing energy costs of such organizations is crucial to advance in these fields. In addition to explicitly reducing energy consumption, it is also possible to cut down the total electricity bill by exploiting spatial and temporal variations in electricity prices.

In the first part of the thesis, we propose a volunteer computing network where peers can set monetary budgets, limiting the financial burden incurred on them due the usage of their computational resources. Assuming that the price of the electricity consumed by the peers has temporal variation, we show that our approach leads to an interesting task allocation problem, where the goal is to maximize the amount of work done by the peers without violating the monetary budget constraints set by the peers. We propose various polynomial time heuristic algorithms to the problem, which is NP-hard, and our extensive simulations show that our approach can increase the total amount of work done up to 35% compared to an existing baseline.

In the second part, we consider a geographically distributed data center network that is specialized to run batch jobs with previously determined Service Level Agreements (SLAs). Taking into account the spatial and temporal variations in the electricity prices and free cooling opportunities by utilizing the outside weather, we model the problem of minimizing the energy cost as a linear programming problem. We propose two job scheduling heuristic algorithms and our simulations using real-life workload traces and electricity prices demonstrate that the proposed heuristics can decrease the total energy cost up to 6% compared to a load balancing baseline solution.

ÖZETÇE

Büyük ölçekli dağıtılmış sistemler güvenilir servis sağlamak ve karmaşık problemlerin çözümü için yüksek miktarlarda hesaplama gücüne ihtiyaç duyarlar. Bu bağlamda, bu tarz sistemlerin enerji ihtiyaçları hızlı bir şekilde artmakta ve bu da beraberinde çok yüksek maliyetler getirmektedir. Bu alanlarda daha ileri noktalara gelebilmek için bu maliyetleri düşürmek bu organizasyonlar için hayati önem taşımaktadır. Kullanılan toplam enerji miktarını düşürmenin yanı sıra, elektrik fiyatlarında görülen coğrafi ve zamana bağlı değişimlerden faydalanarak elektrik faturalarını düşürmek de mümkün olmaktadır.

Tezin ilk kısmında, kullanıcıların finansal bütçeler belirleyebildiği ve bu sayede kendi kaynaklarının kullanımından doğan finansal yükü sınırlayabilecekleri bir gönüllü işlem ağı sunuyoruz. Kullanıcıların tükettikleri elektrik fiyatının zamana bağlı olarak değiştiği varsayımı altında, yaklaşımımızın ilginç bir görev atama problemi oluşturduğunu gösterdik. Burada amaç kullanıcıların belirledikleri bütçelerini aşmayacak şekilde işlem ağında yapılan toplam işin maksimuma taşımak. NP zorlukta olan bu probleme çözüm olarak polinom zamanda çalışan sezgisel algoritmalar sunduk ve detaylı simulasyonlarımız sonucunda gönüllü işlem ağında işlenen toplam iş miktarının şu anda kullanılmakta olan tekniklere oranla %35 arttırılabileceğini gösterdik.

İkinci kısımda, toplu işlerin çözümü için özelleşmiş coğrafi olarak dağıtımlı veri merkezleri modelliyoruz. Elektrik fiyatının mekana ve zamana bağlı olarak değiştiği varsayımını ve dışarıdaki havayı kullanan soğutma fırsatlarını göz önüne alarak, enerji masrafını azaltmayı lineer programlama problemi olarak modelliyoruz. İki sezgisel iş planlama algoritması sunuyoruz ve gerçek sistem kayıtları ve elektrik fiyatlarını kullandığımız simülasyonların sonucuna göre sunduğumuz algoritmalar toplam enerji masraflarını iş dengeleme amaçlı algoritmalara oranla %6'ya kadar azaltmaktadır.

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to my advisor Assoc. Prof. Öznur Özkasap. Her guidance and insights helped me a lot to finalize my research with success. I offer a special gratitude to Dr. Berkant Barla Cambazoğlu, without his persistence help and invaluable contributions this thesis would not have been completed.

I want to thank to my thesis committee members, Assoc. Prof. Yücel Yemez and Prof. A. Murat Tekalp, for their time, effort and valuable remarks.

My heartfelt appreciation goes to my dear friends in Networked and Distributed Systems Lab as well as in Graduate School: Adilet Kachkeev, İrem Nizamoğlu, Seyhan Uçar, Yalçin Sadi and many others, and the sincerest gratitude to my lifelong friends, Burak Özen and Orçun Simsek, whom I shared the best times of my life.

I am particularly grateful for the financial support of Scientific and Technological Research Council of Turkey (TUBITAK) and COST (European Cooperation in Science and Technology) framework, Action IC0804.

I am deeply grateful to my family for their endless support in my entire life. My mother, Gülsüm Güler, is the kindest and the most caring person I will ever know. My father, Nazir Güler, always supported my decisions and thought me to think freely. My beautiful little sister, Hilal Güler and my little cool brother, Ertunç Güler. They are everything for me and they will always be.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Large scale distributed systems have seen growing interests and advances since the emergence of Internet and related Internet services. Complex real world problems, communication and information sharing between hundreds of millions people directed researchers and industrial bodies to find efficient and reliable solutions which involve several different distributed system paradigms, and P2P computing and networking, Grid computing and Cloud computing systems are the best known examples.

In the first part of our work, we focus on a specific class of distributed P2P computing system, Volunteer computing. Volunteer computing became popular in the last decade. In volunteer computing networks, a large number of geographically distributed computer owners (peers) donate their resources (e.g., network, CPU, and storage) for use in a certain large-scale project, whose objective is to solve a computationally expensive problem by harnessing the provided resources. The best known examples of such networks are SETI@Home,[1] Folding@Home,[2] ClimatePrediction,[3] and ABC@Home.[4]

In volunteer computing networks, a central authority is responsible for the management of the network. Typically, the central authority divides a large problem instance into smaller tasks and distributes them among the peers for processing. The peers join the network on a voluntary basis and help the processing without receiv-

---

[1]SETI@Home, `http://setiathome.berkeley.edu/index.php`.

[2]Folding@Home, `http://folding.stanford.edu/`.

[3]ClimatePrediction, `http://climateprediction.net/`.

[4]ABC@Home, `http://abcathome.com/`.

ing any immediate financial benefit. Since the peers allocate their computational resources, which consume energy, they even end up with increased electricity bills. This may create an obstacle for growing the volunteer computing network as peers will be less motivated to join the network.

In some volunteer computing networks, the peers are allowed to specify an upper bound on the number of tasks they are willing to process in a given period of time, or they may specify the time duration their resources can be used [Anderson, 2011, Kondo et al., 2007]. In this work, we go one step beyond and propose an alternative where peers can explicitly specify the maximum amount of money they can afford to spend in a time period while their resources are being used. For example, peers can set monetary limits such as "at most 10 cents per week" or "at most two cents per day". The incentive behind this approach is that allowing such monetary constraints may motivate more peers to contribute to the network since the peers will have a guarantee that the financial overhead incurred by the network will be limited.

From the perspective of the volunteer computing network, having such monetary constraints leads to an interesting task allocation problem. Obviously, the goal of the network is still to maximize the amount of work done in a given time period. However, under the assumption that the electricity prices show temporal variation [Kayaaslan et al., 2011, Qureshi et al., 2009, Rao et al., 2010b], the dispatcher now needs to decide when to assign a task to a peer for processing. Assigning tasks to a peer when the peer is consuming electricity at high prices will lead to quick exhaustion of the peer's monetary budget. Hence, it is important for the dispatcher to estimate the time periods where the peers are consuming cheap electricity and assign the tasks to them accordingly, trying to exhaust peers' monetary budgets as much as possible but without exceeding them.

The second part of the thesis is focused on Cloud computing systems. Serving large amount of tasks and managing big amount of data generated by the Internet services as well as IT industries can be done efficiently with the emerging cloud systems that provide massive computing power. Majority of the Internet services have

started to move to the server side due to growing complexity of these services and the trend from client-side computing to server-side computing directed several key IT companies to build their own massive data centers. For instance; Google, Amazon, Facebook and Microsoft are among the first that operate their own data center networks referred as Warehouse Scale Computers (WSCs) in [Barroso and Hölzle, 2009], since they highly differ from traditional data centers. In general, the WSCs host thousands of computing nodes that are mostly homogeneous, constantly interacting with each other in order to process complex and massive tasks.[5]

A major cost of running such large scale cloud data centers is the energy cost associated with them. Energy consumption of cloud data centers include several factors, and CPU energy consumption and cooling overheads are the most dominant factors (42% being CPU energy usage and 15.4% being the cooling overhead [Barroso and Hölzle, 2009]). Aforementioned big IT companies mostly utilize hundreds of thousands of servers geographically distributed around the world to better serve their customers and such distributed data centers cost millions of dollars to run. Thus, even small reductions achieved in power cost imply huge cost savings [Yao et al., 2012]. For example, a recent Google report states they saved over one billion dollars to date from their energy efficiency efforts.[6] Moreover, data centers' high computing power also require complex and advanced cooling infrastructures to cool down heated servers and network components, and cooling cost associated with the operation of a data center is one of the biggest consumers of the total electricity used [Barroso and Hölzle, 2009]. Airside economization systems offer some opportunities to use cold and dry outside weather in order to lower the power usage efficiency (PUE) of a particular data center.[7]

Decreasing the energy consumption of geographically distributed data centers has been an active area of research. However, in addition to improving energy efficiency,

[5]We refer to WSCs as cloud data centers in the rest of the the thesis.

[6]Google green, The Big Picture:`http://www.google.com/green/bigpicture`

[7]`http://www.datacenterknowledge.com/archives/2013/08/08/building-efficient-data-centers/`

there is also an opportunity to decrease the total electricity bill of a data center by considering temporal and spatial variations in electricity prices. In particular, geographically distributed cloud data centers can greatly benefit from these variations by scheduling their tasks to and allocating their resources in electrically cheaper places. Thus, optimizations considering temporal and spatial variation in electricity prices can achieve reductions in the total electricity bill of data centers.

In general, cloud data centers are responsible for computing different set of jobs including batch jobs, interactive web queries, big data analysis and many others which have different characteristics and requirements. These requirements have direct impact on setting business goals which are mostly conflicting in its very sense. For example, in web search queries, response time is the most important objective and it needs to be kept under several milliseconds [Kayaaslan et al., 2011]. On the other hand, for big data analysis or batch jobs, the objective is maximizing the throughput or reducing the computation time. In this work, we focus on delay tolerant batch jobs to better observe the effects of time related changes in data centers, in other words, having the ability to delay some jobs to better exploit temporal electricity price variations. However, service providers are entitled to satisfy certain requirements detailed in Service Level Agreements (SLAs), thus it is not always possible to delay each job as much as desired. Any violation in SLAs, for example missing a job's deadline, results in paying penalty fee that brings additional costs.

Required massive computing power in data centers and its related cost push service providers to constantly chase novel methods to reduce their energy cost, and reducing the energy consumption of large-scale distributed systems has recently been a hot research topic. Some studies focus on server consolidation by moving virtual machines across data centers and also consider SLA penalties if the deadline of a certain job is not satisfied [Buchbinder et al., 2011, Li et al., 2012]. Le et al. [Le et al., 2010] and Gao et al. [Gao et al., 2012] concentrate on the greenness aspect of data centers and allow service providers to trade off between the electricity cost and the carbon footprint. Some studies investigate the data center cooling problem and pro-

pose solutions based on workload placement [Moore et al., 2005, Tang et al., 2008]. In recent years, researchers have also investigated the impact of spatio-temporal electricity price variations on financial cost savings [Qureshi et al., 2009, Ren et al., 2012, Zhang et al., 2012].

Our focus in this part, however, is not to decrease energy usage but to minimize the total electricity cost of a geographically distributed data center system by scheduling incoming batch workload jobs. Above mentioned individual aspects (computational cost, cooling cost and penalty cost) that constitute the total financial cost of a cloud data center are tried to be solved in isolation. However, to the best of our knowledge, no prior work presented a complete cost model incorporating all of the issues.

1.1 Contributions

Contributions of this thesis can be given under two main topics. First is the contributions of our work of task allocation in volunteer computing networks:

- We investigate the problem of allocating tasks in a volunteer computing network under monetary budget constraints that are set by the peers.
- We formally state the problem and propose various heuristic solutions.
- We investigate the performance of the proposed heuristics through simulations based on realistic data traces and real-life electricity prices.
- The empirical results indicate significant improvements (up to 30%) in the task processing capacity of the network relative to an existing baseline.

Second, we list the contributions of our work in energy cost optimization in distributed cloud data centers:

- We introduce a detailed cost model for geographically distributed cloud data centers by including IT related server costs, cooling costs and penalties due to any SLA violation.

- We propose two job scheduling heuristics as solution that exploit spatial and temporal electricity price differences between data centers and 'free cooling' opportunities by utilizing outside weather.

- We conduct extended experiments in a realistic setting (with real world workloads and real electricity prices) using a detailed simulator.

- Our experimental results show that total energy cost of such large scale systems can be reduced up to 6% using our proposed algorithms depending on SLA constraints.

1.2 Organization

The rest of the thesis is organized as follows. In Chapter 2, we survey the related literature in two parts; first is a review of works in volunteer computing domain and the second is a literature review of energy optimization efforts in large scale distributed systems. Chapter 3 provides the study of task allocation in volunteer computing networks under monetary budget constraints, with problem formulation, simulation setup and experimental results. Similarly, in Chapter 4, we investigate energy cost aware job scheduling in geographically distributed cloud data centers with a complete cost model, heuristic solutions, simulation environment and performance results. The thesis is concluded in Chapter 5.

Chapter 2

RELATED WORK

We provide the literature review in two groups. First group of work is related to volunteer computing domain and the second group focuses on energy cost optimization in distributed data centers.

2.1 Volunteer Computing

The dynamics of volunteer computing systems are explained in [Kondo et al., 2007, Anderson, 2007], focusing on the BOINC middleware system, which provides local schedulers exploiting resources of the participating hosts such as CPU and memory resources based on host availability. The schedulers take into account job specific properties like the length and deadline of the job. This system employs user constraints for CPU availability as we employed, but omits the financial costs incurred on the peers. In our problem, the focus is on the total computational work done by the peers in the network within given budget constraints.

Estrada et al. [Estrada et al., 2006, Taufer et al., 2007] focus on the availability and reliability of peer resources in order to determine how to distribute the tasks to existing peers. To this end, they introduce some threshold-based heuristics. They simulate their heuristics using the SimBA simulator and compare against a naive baseline. Different from our local task allocation heuristics, their heuristics work on the server-side of the system.

There are also some studies and real-life projects focusing on voluntary cloud computing [Costa et al., 2011, Kondo et al., 2009, Lombraña González et al., 2012, Himyr et al., 2012]. Basically two sustainability models for voluntary cloud computing paradigm are used: a non-profit volunteer cloud and a commercial volunteer

cloud, in which a charity engine model is introduced. The commercial voluntary cloud includes selling volunteer resources in exchange of some monetary means to other entities in the cloud (e.g., one cent per core-hour resource).[1] Even though some lottery systems are proposed for keeping the existing peers in the system, giving the peers the option to set monetary budgets on their resources may provide an extra level of motivation.

2.2 Energy Cost Optimization

Reducing the energy consumption of large-scale distributed systems has been recently a hot research topic [Barroso and Hölzle, 2009]. A comprehensive comparison of previous works is listed in Table 2.1. We extracted some identifying features that are important in optimizing energy consumption and/or energy bill of cloud data centers. These features are:

- Problem: Is the problem online or offline? The problem type significantly affects how it is handled.

- Techniques: There are different techniques to follow and the table is grouped according to these techniques. Abbreviations used for these techniques are as follows: WS - Workload Scheduling, SOIS - switching off idle servers, CFS - CPU frequency scaling, DVFS - dynamic voltage frequency scaling, BCD - battery charge/discharge, SC - server consolidation, TS - thermal storage, OS - Server Outsourcing.

- Electricity price: Is spatial or temporal electricity price variation exploited?

- Cooling: Is the cooling cost considered?

- Temperature: Is temperature variation taken into account during the cooling process?

[1] Charity Engine, `http://www.charityengine.com/`.

- SLA/Penalty: Are service level agreements utilized during problem solving, especially penalty terms?
- Workload: What type of workload is used for testing proposed solutions?
- Greenness: Is the total energy consumption (not the cost) minimized?

Some recent works try to reduce the total electricity bill of data centers by taking into account the spatial and/or temporal variation in the electricity prices [Liu et al., 2011, Sakamoto et al., 2012, Xu et al., 2013, Buchbinder et al., 2011] and [Garg et al., 2011b, Zhang et al., 2012, Mani and Rao, 2011, He et al., 2012] and [Yigitbasi et al., 2011, Sadhasivam et al., 2009, Van den Bossche et al., 2010]. Qureshi et al. [Qureshi et al., 2009] is the first to propose exploiting electricity price variations in order to minimize energy costs in distributed data centers. They utilize workloads obtained from Akamai, a content delivery network, to perform simulations that illustrate potential financial gains. Ren et. al. [Ren et al., 2012] propose an efficient online scheduling algorithm named GreFar for batch jobs in geographically distributed data centers. They consider the metrics of energy cost and fairness, and achieve energy cost savings at the expense of fairness among users. Gao et. al. [Gao et al., 2012] propose a dynamic workload control algorithm in order to adjust the trade-off between access latency, carbon footprint and electricity cost according to changes in request load and carbon footprint. They also propose an upgrade plan for the data centers due to growth in request loads. Rao et. al. [Rao et al., 2010b] follow a similar approach to work of [Qureshi et al., 2009] but they exploit a multi-regional electricity market rather than wholesale market with the same objective using Brenners algorithm. Yao et. al. [Yao et al., 2012] construct a general framework for power cost reduction by exploiting time and space variations in electricity prices. They work in two time scales; one is for global scheduling of workloads and the other is for in-data center scheduling. In [Sankaranarayanan et al., 2011], they investigate heterogeneous data centers that are globally distributed and propose greedy heuristic solutions for global and local scheduling of incoming requests

with the objective of minimizing energy cost using electricity price variations. Le et al. [Le et al., 2009] and [Le et al., 2010] propose optimization based request distribution framework aiming to either reduce energy cost by using electricity price variations or reduce the brown energy consumption in the expense of acceptable cost increases by dispatching the incoming requests to data centers located near green energy resources. In [Guo et al., 2011] and [Urgaonkar et al., 2011], the authors investigate cost reduction by using uninterrupted power supply (UPS) units as energy storage devices in which they charge the UPS units when electricity is cheap and use the stored energy instead of regular grid when electricity price is high.

Techniques of dynamic voltage frequency scaling, turning off idle servers and server consolidation have been proposed to reduce the total energy consumption of cloud data centers and some of these works also consider the cooling related energy costs [Rao et al., 2010a, Lin and Ng, 2005, Young et al., 2013, Krioukov et al., 2010, Lin et al., 2011, Chase et al., 2001, Zhang et al., 2013, Mazzucco and Dyachuk, 2012b, Mazzucco and Dyachuk, 2012a, Stanojevic and Shorten, 2010] and [Wang et al., 2013, Garg et al., 2011a, Goiri et al., 2012]. Leon et al. [León and Navarro, 2011] aim to reduce the energy consumption of the allocation of resources to networked applications by proposing an algorithm that finds an upper bound on energy saving by optimizing the energy consumption as well as resource requirements of applications. In [Xu and Liu, 2012], they design an intelligent through filling system for delay tolerant jobs in order to achieve cost savings by obeying delay constraints. Dynamic speed scaling, electricity price diversity and statistical multiplexing are the key contributions of their energy cost efficiency efforts. Li et al. [Li et al., 2012] mathematically formulate the problem of dispatching job requests to geographically distributed data centers in order to minimize the overall energy cost. They consider examining temperature effect on the cooling cost in addition to temporal and spatial electricity price variations. However, they solve the problem only by means of LP solver like CPLEX for very small cases (three data centers each having only three servers) and do not propose any solution that can be applied to realistic cases. Shah and

Krishnan [Shah and Krishnan, 2008] show that, in a globally connected data center network, dynamic optimization of the thermal workloads based on local weather patterns can reduce the environmental burden by up to 30%.

In addition to studies examining variations in electricity prices, there exist some works [Moore et al., 2005, Pakbaznia and Pedram, 2009, Wang et al., 2011] and [Weerts et al., 2012] that aim at reducing energy consumption by enhancing cooling units and their placement. They are making use of economization systems in which they try to exploit as much free resources (air, water, etc.) from the environment as possible to reduce the cost of cooling efforts.

Table 2.1: Literature Review Comparison

Reference	Problem	Techniques	Electricity price	Cooling	Temperature	SLA / Penalty	Workload	Greenness
[Gao et al., 2012]	Online	WS	Spatial, temporal	–	–	–	Web	–
[Ren et al., 2012]	Online	WS	Spatial, temporal	–	–	–	Batch	–
[Rao et al., 2010b]	Offline	WS	Spatial, temporal	–	–	+	Web	–
[Liu et al., 2011]	Offline	WS	Spatial, temporal	–	–	–	Web	+
[Yao et al., 2012]	Online	WS	Spatial, temporal	–	–	–	Batch	–
[Sakamoto et al., 2012]	Online	WS	Spatial, temporal	–	–	–	Web	–
[Le et al., 2009]	Offline	WS	Temporal	–	–	–	Web	+
[Sankaranarayanan et al., 2011]	Online	WS	Spatial, temporal	–	–	–	Web	–
[Xu et al., 2013]	Offline	WS	Spatial, temporal	+	+	–	Batch, Web	–
[Buchbinder et al., 2011]	Online	WS	Spatial, temporal	–	–	–	Batch	–
[Le et al., 2010]	Online, Offline	WS	Spatial, temporal	–	–	+	Web	+
[Garg et al., 2011b]	Online	WS	Spatial	–	–	+	Batch	+
[Qureshi et al., 2009]	Offline	WS	Spatial, temporal	–	–	+	Web	–
[Zhang et al., 2012]	Offline	WS	Spatial, temporal	+	–	+	Web	–
[Mani and Rao, 2011]	Online	WS	Spatial, temporal	–	–	+	Web	–
[He et al., 2012]	Online, Offline	WS	Spatial, temporal	–	–	+	Random	+
[Yigitbasi et al., 2011]	Online	WS	–	–	–	–	Batch, Web	+
[Sadhasivam et al., 2009]	Online	WS	–	–	–	+	Batch, Web	–
[Van den Bossche et al., 2010]	Offline	WS	–	–	–	+	Batch	–
[Xu and Liu, 2012]	Offline	WS, CFS	Spatial, temporal	–	–	+	Batch	–
[Rao et al., 2010a]	Online	WS, CFS, SOIS	Spatial, temporal	–	–	–	Web	–
[Urgaonkar et al., 2011]	Online	BCD	Spatial, temporal	–	–	–	Random	–
[Guo et al., 2011]	Offline	BCD	Spatial, temporal	–	–	–	Theoretical	–
[Li et al., 2012]	Offline	WS, CFS, SOIS	Spatial, temporal	+	+	–	Web	–
[Lin and Ng, 2005]	Offline	WS, DVFS	–	–	–	+	Random	+
[Young et al., 2013]	Online	DVFS	–	–	–	+	Batch	–
[Krioukov et al., 2010]	Online	WS, SOIS	–	–	–	–	Web	+
[Lin et al., 2011]	Online	SOIS	–	–	–	–	Web	–
[Chase et al., 2001]	Online	SOIS	–	–	–	+	Web	–
[Zhang et al., 2013]	Offline	SOIS	–	–	–	–	Batch, Web	+
[Mazzucco and Dyachuk, 2012b]	Online	SOIS	–	+	–	+	Batch, Web	+
[Mazzucco and Dyachuk, 2012a]	Online	SOIS	–	+	–	–	Web	+
[Stanojevic and Shorten, 2010]	Offline	CFS	Temporal	–	–	–	Random	–
[Wang et al., 2013]	Online	SC	Spatial, temporal	–	–	+	No Info	–
[Garg et al., 2011a]	Online	SC	–	–	–	+	Batch, Web	–
[Goiri et al., 2012]	Online	SC, OS	–	–	–	+	Batch, Web	–
[Shah and Krishnan, 2008]	Online, Offline	WS	Spatial	+	+	–	Random	+
[Moore et al., 2005]	Online	WS	–	+	+	–	Batch	–
[Pakbaznia and Pedram, 2009]	Online	SC	–	+	–	–	No info	+
[Wang et al., 2011]	Online	TS	Spatial, temporal	+	–	–	Random	–
[Weerts et al., 2012]	Online	SC	–	+	–	–	Real-time	+

Chapter 3

TASK ALLOCATION IN VOLUNTEER COMPUTING NETWORKS UNDER MONETARY BUDGET CONSTRAINTS

In this chapter, we first formally state the investigated task allocation problem in volunteer computing networks and propose heuristic solutions. Then, we explain our simulation setup and provide the comparative performance results of our proposed algorithms.

3.1 Problem Specification

We consider a volunteer computing network consisting of N peers and a central dispatcher. The dispatcher can exploit the computational resources of the peers to perform certain tasks. For example, some data can be transferred from the dispatcher to a peer for processing, and the peer may return some output data that it produced back to the dispatcher. We assume that the main computational overhead is in the processing of the tasks by the peers and the communication overheads are negligible since this is typically the case in practice. In our problem specification, we also omit the financial costs potentially incurred by Internet service providers due to the allocated network bandwidth.

Each peer i, when joining the network for the first time, specifies a personal monetary budget B_i, which indicates the maximum electricity bill increase that the peer can afford while it contributes to the processing of the tasks assigned by the dispatcher. This budget forms an upper bound on the monetary cost the network is allowed to incur on the peer due to the exploitation of the computational resources

of the peer. The budgets are defined over time periods, each with a fixed length of T units of time.[1] The dispatcher keeps track of an estimate of the monetary cost incurred on each peer during the current time period. The cost estimates are reset to zero at the end of every time period. The computational resources of a peer can be used only if its monetary budget is not yet fully consumed within the current time period. For example, assuming weekly sessions starting on Monday, if the budget of a peer is reached on Friday, the peer's resources cannot be exploited in the remaining two days of the week. However, the peer becomes available again by the start of the next week.

To simplify the problem definition, we assume that the time is partitioned into unit time intervals, i.e., the time is not continuous but discrete. More specifically, the time period T is divided into T/u time slots, each of length u time unit. The dispatcher can allocate a time slot $[t, t+u)$ either entirely or may decide not to allocate it at all. A peer can process the tasks assigned to itself independent of the other peers and tasks. Therefore, the allocation problem we will describe can be optimized separately for each peer.

When subscribing to the network, the peers let the dispatcher know about the properties of their computational resources (e.g., the number of cores and the CPU clock frequency).[2] Also, when joining the network, the peers independently set a minimum and a maximum CPU utilization threshold, which are denoted by m_i and M_i, respectively. Moreover, at the beginning of every time slot $[t, t+u)$, through a locally running daemon, each peer i lets the dispatcher know about its expected CPU utilization $U_i(t)$ in that time slot. The CPU utilization $\widehat{U}_i(t)$ due to the tasks assigned by the dispatcher is bounded by $\max(M_i - U_i(t), 0)$, i.e., the dispatcher fully utilizes the CPU of the peer without exceeding M_i. The tasks can be allocated on the computational resources of peer i only if $U_i(t) < m_i$, i.e., the expected CPU utilization in the given time slot must be less than the minimum utilization threshold

[1] In our work, we assume that the budgets are set on a weekly basis. Some of the solutions presented in Section 3.2 will be based on this assumption.

[2] We assume that each peer has a single CPU with varying clock frequencies.

value. Otherwise, the peer's resources are not used since the peer is assumed to be actively using them. Under these constraints, we have

$$\widehat{U}_i(t) = \begin{cases} M_i - U_i(t), & m_i < U_i(t) \leq M_i, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

We also assume that the dispatcher has access to the information about the current and historical electricity prices for each peer and exploits the fact that the electricity prices show temporal variation [Kayaaslan et al., 2011, Qureshi et al., 2009]. The dispatcher, however, has no knowledge of the future electricity prices. We denote by $E_i(t)$ the price of the electricity for peer i in time slot $[t, t+u)$. If the computational resources of peer i are used by the network during the time slot $[t, t+u)$, the increase $I_i(t)$ in the electricity bill of peer i is estimated by

$$I_i(t) = \widehat{U}_i(t) \times W_i \times E_i(t) \times u, \tag{3.2}$$

where W_i denotes the power consumption of peer i's CPU in watts.

For a given peer and a particular time slot, the utility is measured by the computational work done by the peer in that time slot. The computational work is defined in terms of the clock frequency of the peer and the expected CPU utilization due to execution of the tasks issued by the dispatcher in the given time slot. More formally, the computational work done by peer i in time slot $[t, t+u)$ is denoted by $J_i(t)$ and defined as

$$J_i(t) = \widehat{U}_i(t) \times F_i, \tag{3.3}$$

where F_i denotes the CPU clock frequency of peer i.

Given the above-mentioned definitions and notations, which are summarized in Table 3.1, our objective is to maximize

$$\sum_{i=1}^{N} \sum_{t=0}^{\lfloor T/u \rfloor} J_i(t), \tag{3.4}$$

subject to

$$\sum_{t=0}^{\lfloor T/u \rfloor} I_i(t) \leq B_i, \quad \text{for } 1 \leq i \leq N. \tag{3.5}$$

Table 3.1: System parameters

Parameters	Symbol
Number of peers in the network	N
Monetary budget of peer i	B_i
Time period after the monetary budgets of peers are reset	T
Length of a unit time slot	u
Increase in the electricity bill of peer i in time slot $[t, t+u)$	$I_i(t)$
Electricity price for peer i in time slot $[t, t+u)$	$E_i(t)$
Expected CPU utilization of peer i in time slot $[t, t+u)$	$U_i(t)$
Maximum CPU utilization threshold for peer i	M_i
Minimum CPU utilization threshold for peer i	m_i
CPU clock frequency of peer i	F_i
Power consumption of peer i's CPU in watts	W_i

In other words, the goal is to maximize the total amount of computational work done in a given time period while respecting the monetary budget constraints set by the peers.

3.2 Solutions

In this section, in order to simplify the presentation, we refer to the time slots as tasks. This is because, in our problem definition, the time is discretized into time slots and each time slot is either fully allocated for processing tasks or not allocated by the dispatcher. We assume that the dispatcher has infinitely many tasks. The goal is to determine in which time slots to allocate the computational resources of each peer, rather than how to schedule individual computational tasks for execution.

We note that, if perfect knowledge of future electricity prices is available, our task allocation problem can be formulated as the 0-1 knapsack problem. In this formulation, the knapsack value corresponds to the monetary budget of the peer and

the items correspond to the tasks (time slots). An item's weight corresponds to the cost of allocating the respective task, i.e., occupying the resources of the peer during the respective time slot (see Eq. 3.2). The value of an item is determined by the work done in the respective time slot (see Eq. 3.3).

In our scenario, naturally, the dispatcher has no information about the future electricity prices. The time slots must be allocated on-the-fly using only the past electricity price information. Consequently, our problem reduces to the online knapsack problem, where the item weights and values are not known beforehand. In our case, we also have an additional peer availability constraint.

The online knapsack problem is first studied in [Marchetti-Spaccamela and Vercellis, 1995] and its solution is applied to different problems, including the auction bidding problem [Zhou et al., 2008, Zhou and Naroditskiy, 2008] and the knapsack secretary problem [Babaioff et al., 2007]. The problem is known to be NP-hard [Babaioff et al., 2007] and hence there is no polynomial-time algorithm for an optimal solution. Since the online knapsack problem is NP-hard, our task allocation problem is also NP-hard. Hence, heuristics play an important role in finding solutions. In the rest of the section, we present some heuristic solutions for our problem.

3.2.1 Algorithms

Naive Baseline (`Baseline`)

This approach does not take into account the temporal variation in the electricity prices. A time slot is allocated by the dispatcher if the peer's computational resources are available in that time slot, i.e., whenever the peer is online and the minimum CPU utilization constraint set by the peer is satisfied (i.e., $U_i(t) < m_i$ must hold). In this approach, since the time slots are allocated without considering the price of the electricity, there is the risk of using the peers' computational resources when the electricity price is high. Consequently, the monetary budgets of the peers may be quickly exhausted, potentially reducing the amount of work done by the peers.

Algorithm 1 A generic algorithm for the heuristic solutions that rely on the expected electricity price computed using the past electricity price information.

```
totalWork ← 0
t ← 0
while t < T do
  for i = 0 to N do
    expectedPrice ← calculateExpectedPrice(i, t)
    if E_i(t) < expectedPrice then
      if I_i(t) ≤ B_i then
        if U_i(t) < m_i(t) then
          totalWork ← totalWork + Û_i(t) × F_i
          B_i ← B_i − I_i(t)
    i ← i + 1
  t ← t + u
return totalWork
```

Solutions Based on the Expected Electricity Price

The main idea behind this class of heuristics is to exploit the past electricity prices to decide whether the price of the electricity currently consumed by the peer is relatively high or not. The techniques compute an expected electricity price value based on the price data observed in the past. To decide whether the current time slot should be allocated for processing tasks, the electricity price estimated for the current time slot is compared against the expected electricity price value. If the current electricity price is lower than that value, the time slot is allocated; otherwise, it is not allocated. A generic algorithm for this class of heuristics is provided in Algorithm 1. We summarize below three alternative approaches for computing the expected electricity price.

Average of Yesterday (`Yesterday`): The expected electricity price is computed by averaging the sample price values observed on the previous day. For example, if the current day is Wednesday, the expected price is computed by taking the average of

price values sampled from Tuesday. This technique exploits the fact that prices do not differ too much between consecutive days, i.e., it exploits the temporal locality in electricity prices.

Past Average of Today (`SameDayHistory`): The expected electricity price is computed as the average of the electricity prices observed on the previous occurrences of the current day of the week. For example, if the current day is Wednesday, the average electricity price of all Wednesdays in the past is used as the expected price. The rationale behind this technique is that the electricity prices tend to be similar on the same day of the week, i.e., it exploits the weekly periodicity of prices.

Average of Entire History (`EntireHistory`): In this technique, the expected electricity price is computed as the average price value over the entire price history. This technique is based on the expectation that all data available about the past electricity prices would be useful.

Since the allocation decisions are made in run-time based on a simple comparison operation, the running time complexity of the above-mentioned heuristics is $O(1)$. Herein, we also briefly elaborate on the availability of competitive bounds. Our heuristics are deterministic online algorithms, which are often compared with their offline versions to define competitive bounds. However, in its general form, there is no competitive deterministic algorithm for the online knapsack problem [Böckenhauer et al., 2012]. Hence, it is not possible to prove a competitive ratio for our heuristics.

History Repeats (`HistoryRepeats`)

This heuristic tries to exploit the weekly repetition in the electricity prices. We compress the entire price data into a single week of data by computing an average value for each hour in a week. The assumption is that the electricity prices of the current week will be identical to these aggregate price values. Under this assumption, the remaining time slots of the current week are stored in a candidate list in increasing order of their estimated electricity prices. The candidate list is traversed starting from the time slot with the lowest price towards the time slot with the highest price.

Algorithm 2 Algorithm for the `HistoryRepeats` heuristic.

$totalWork \leftarrow 0$
$t \leftarrow 0$
$period \leftarrow determinePeriod()$
$sortLastWeekElectricityPrice()$
while $t < T$ **do**
  for $i = 0$ to N **do**
    if T mod $period$ **then**
      $candidateList \leftarrow updateCandidateList(i, t)$
    if $(i, t) \in candidateList$ **then**
      if $I_i(t) < B_i$ **then**
        if $U_i(t) < m_i(t)$ **then**
          $totalWork \leftarrow totalWork + \widehat{U}_i(t) \times F_i$
          $B_i \leftarrow B_i - I_i(t)$
    $i \leftarrow i + 1$
  $t \leftarrow t + u$
return $totalWork$

At each step in the traversal, if the cost of the traversed time slot is less than the remaining budget, it is marked to indicate its suitability in terms of the electricity price and its cost is subtracted from the remaining budget. The traversal stops when a time slot whose cost exceeds the remaining budget is encountered. Later, when making allocation decisions, the current time slot is allocated for running tasks only if it is among the marked slots in the candidate list. The candidate list is periodically updated since it becomes outdated in time. The above-mentioned procedure is illustrated in Algorithm 2. Updating the candidate list, in the worst case, requires sorting a list of T/u items. All online allocation decisions between two consecutive updates can be made in constant time.

Online Knapsack (`OnlineKnapsack`)

In this solution, we adapt the algorithm proposed in [Zhou et al., 2008], which has two assumptions about the input data: i) the weight of each item is much smaller than the capacity of the knapsack, i.e., $I_i(t) \ll B_i$, and ii) the value/weight ratio of each item is bounded both from above and below, i.e., $L \leq J_i(t)/I_i(t) \leq U$, for all t. These assumptions enable the proposed algorithm to have a constant competitive ratio of $\ln(U/L)+1$. In our case, we also have the peer availability constraint given in Eq. 3.5. To capture this constraint, we modify the original algorithm such that the U and L values are used along with the remaining budget in order to determine a threshold value. The decision of allocating the current time slot is made based on a comparison between the gain ratio in the current time slot and the threshold value calculated by the function given in [Zhou et al., 2008]. The algorithm allows the system to aggressively allocate time slots at the beginning while most of the budget is available. The system becomes more selective in time as the remaining budget gets smaller.

Oracle (`Oracle`)

In order to set an upper bound on the performance of the proposed heuristics, we design an `Oracle` algorithm that has access to the future electricity prices and the future CPU utilization values of the peers. As explained before, the optimum solution of our problem cannot be found in polynomial time. However, since `Oracle` is assumed to have access to the future electricity prices, the problem is transformed into the traditional 0-1 knapsack problem, for which there is a pseudo-polynomial algorithm that finds an optimum solution.

3.3 Simulation Setup

To evaluate the performance of the proposed heuristics, we simulate a volunteer computing network. The simulator and the heuristics are implemented in C++. In the

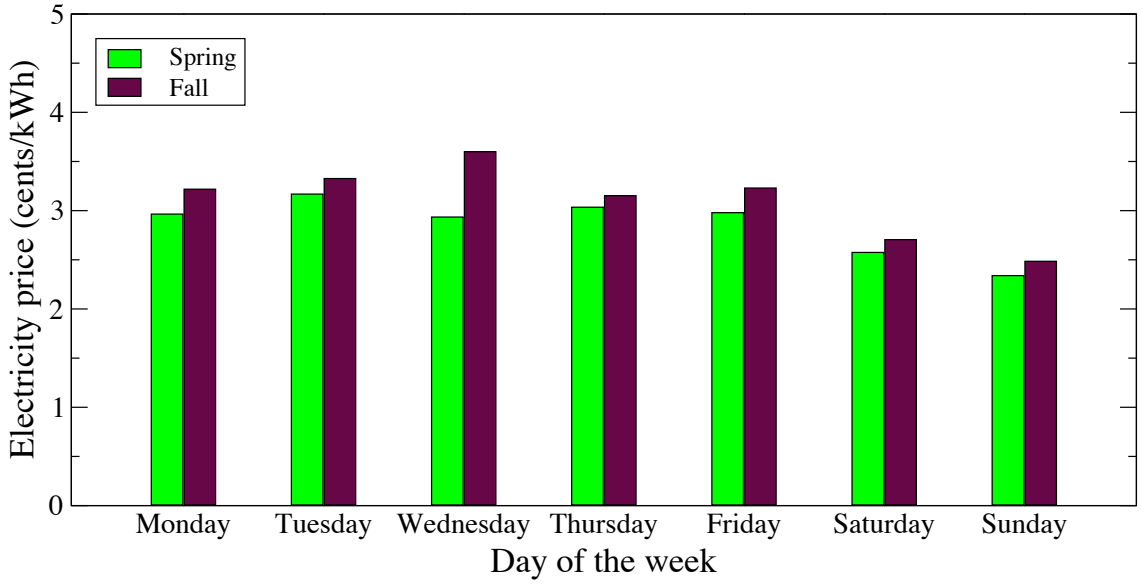

Figure 3.1: Daily change in the average electricity prices for spring and fall.

rest of the section, we provide the details of our simulation.

3.3.1 Electricity Prices

In our simulations, we use real-time electricity prices obtained from ComEd,[3] an electricity provider located in the USA. We collected electricity prices from two different seasons in 2012 (spring and fall) to evaluate the relative performance of each heuristic under varying electricity price data. In particular, the electricity prices are sampled on an hourly basis over a period from 2 April 2012 to 1 July 2012 for the spring season and from 3 September 2012 to 2 December 2012 for the fall season. The last week of each season is used for evaluation while earlier weeks provide the historical price data for some of the heuristics. In Fig. 3.1, we display the average electricity price of each days in a week in our three-month data set. According to the figure, the prices are somewhat correlated between spring and fall, the former having slightly lower prices.

We assume that the peers are located in six different countries: China, Germany,

[3] `https://www.comed.com/customer-service/rates-pricing/real-time-pricing/Pages/rate-besh-pricing-tool.aspx`

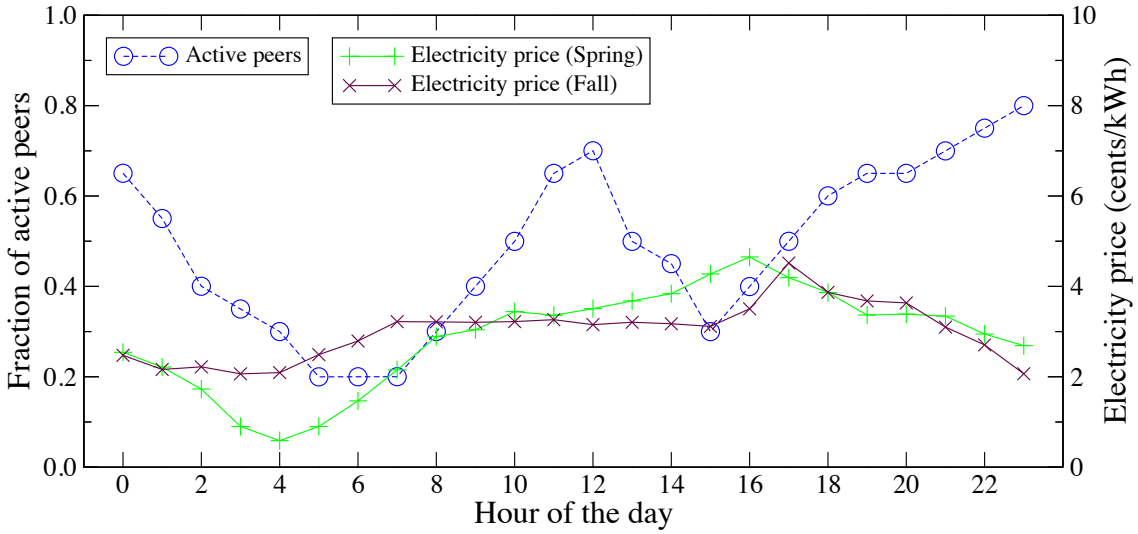

Figure 3.2: Hourly change in electricity prices and the fraction of active peers.

Russia, Turkey, the UK, and the USA. Since the obtained electricity price distribution is representative only for the USA, for each of the remaining countries, we linearly scale this distribution by using the average electricity price in that country. The country-specific average electricity price values are obtained from the Europe's Energy Portal[4] and U.S. Energy Information Administration.[5] Based on the average price of the electricity, the selected countries can be sorted in decreasing price order as Germany, the UK, China, Turkey, the USA, and Russia.

3.3.2 Peers

We assume a volunteer computing network consisting of 10,000 peers. The peers are assumed to be distributed in the previously mentioned countries such that each country receives peers proportional to its presence in the Internet.[6] Hence, in our simulations, we use 5,000, 800, 700, 450, 550, and 2,500 peers from China, Germany, Russia, Turkey, the UK, and the USA, respectively. The minimum and maximum

[4] `http://www.energy.eu/`

[5] `http://www.eia.gov/countries/prices/electricity_households.cfm`

[6] `http://www.internetworldstats.com/top20.htm`

CPU utilization constraints (i.e., the m_i and M_i values) of the peers are selected from a uniform distribution within a range of 15%–20% and 60%–75%, respectively. The CPU clock frequency values (F_i) are sampled from a uniform distribution in the range of 1.7 GHz and 3.2 GHz. The power consumption values (W_i) are taken from the website of Intel, taking into account the clock frequencies.[7]

We also simulate the peers' availability in the volunteer computing network. We assume that this mainly depends on the time of the day. To this end, we rely on the measurements provided in [Bhagwan et al., 2003], where the peer availability is shown to have a diurnal pattern. In Fig. 3.2, we display the hourly change in the electricity prices both in spring and fall seasons together with the fraction of peers that are online. According to the figure, the number of peers makes two separate peaks while the electricity prices have a single peak. We observe a negative correlation in night time: the number of online peers increases while the electricity price decreases until the midnight.

3.3.3 Budgets

An important parameter in our problem is the monetary budgets set by the peers. In practice, the budgets may be affected by many factors, the socio-economic and cultural factors being the most prominent. In this work, since there is no prior published finding in our context, we performed a small-scale user study to determine a reasonable range for the budget values. The study involved 100 participants from Koc University, including undergraduate students, graduate students, and some faculty members. First, we briefly explained the participants what a volunteer computing network is and how it operates using SETI@Home as an example. Then, the participants were asked how much weekly budget they would be willing to allocate voluntarily if they were to join SETI@Home network. We provided them a range of budget options between one cent and 75 cents (the latter is approximately the cost in case the system utilizes a peer's resources in every available time slot). As a result of this study, we decided to

[7] `http://ark.intel.com/`

vary the budget values according to a normal distribution with mean values varying between three and ten U.S. cents. If it is not stated otherwise, the budget values are set to five cents.

3.3.4 Other Parameters and Performance Metrics

We assume that the start of the week is Monday. As the length of a unit time slot (u), we use one minute. In the simulation of the `HistoryRepeats` heuristic, we update the candidate list every six hours.

All reported results are averages of ten runs. As the primary performance metric, we report the total amount of computational work done by the peers (see Eq. 3.3). As secondary metrics, we report the fraction of peers whose budgets are consumed less than a certain threshold and the total electricity bill incurred by the volunteer computing network.

3.4 Experimental Results

In Fig. 3.3 and Fig. 3.4 we compare the performance of our heuristics in terms of the amount of work done by the peers for varying monetary budget values. As expected, all of the proposed heuristics perform better than `Baseline` and worse than `Oracle`. On average, the heuristics relying on comparisons with expected electricity price (`Yesterday`, `SameDayHistory`, and `EntireHistory`) achieve slightly better performance than the `Online Knapsack` and `HistoryRepeats` algorithms for both seasons.

According to Fig. 3.3, `SameDayHistory` is the best performing heuristic. For the spring season, the total computational work yield by this heuristic is 19% higher than that yield by the `Baseline` and 14% less than that of `Oracle`, on average. According to Fig. 3.4, for the fall season, `Yesterday` provides the best performance with 20% higher computational work compared to `Baseline` and 8% less work compared to `Oracle`, on average. Moreover, we observe that the total amount of computational work done by each algorithm in the spring season is higher than the work done in the

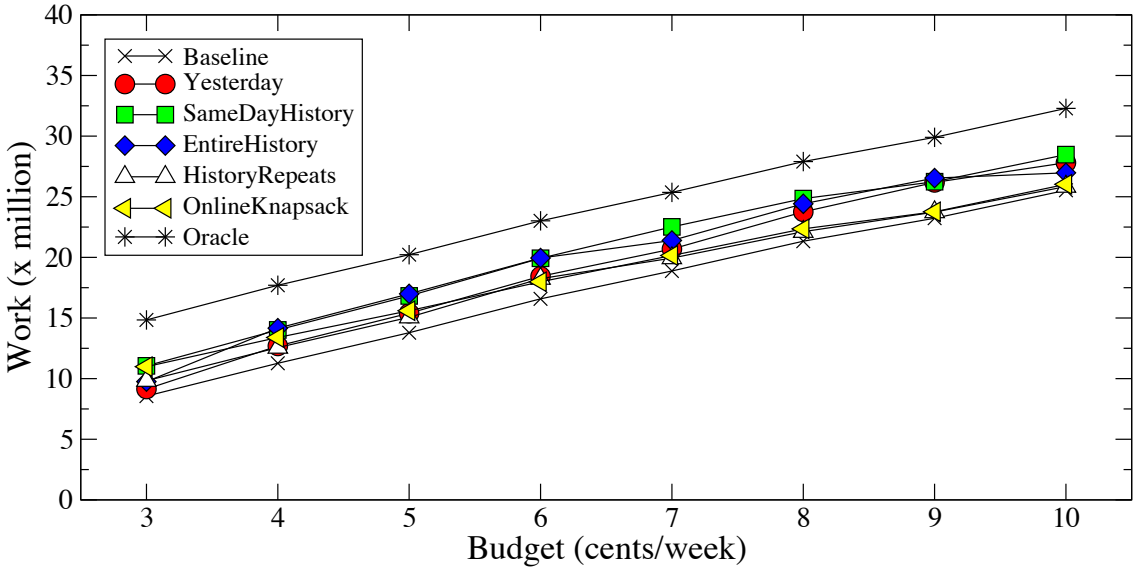

Figure 3.3: Computational work distribution with respect to weekly budgets in Spring

fall season. This is due to the relatively lower electricity prices in the spring season (see Fig. 3.1). These reported performance improvements are average values over all budget scenarios. In fact, the performance of our heuristics tends to be better when lower budget values are used. For example, when the mean budget value is set to three cents, in fall season, the `Yesterday` heuristic improves over the total computational work in case of `Baseline` by 34% and it is only 9% lower than the optimum solution.

As the budget values get higher, the performance gap between the proposed heuristics and the baseline starts to diminish. This is because higher budget values imply more flexibility in allocating the time slots while heuristics still limit them from being allocated. In other words, it becomes less important to allocate time slots in which the electricity prices are low. In an extreme case, if we set the budget values to infinity, `Baseline` would attain the optimum result since it would allocate all available slots without any budget constraint. However, this does not necessarily mean that the performance of the proposed heuristics decreases with higher budget values because, while the performance of `Baseline` improves relative to the heuristics, in the meantime, the performance gap between the heuristics and the optimum solution

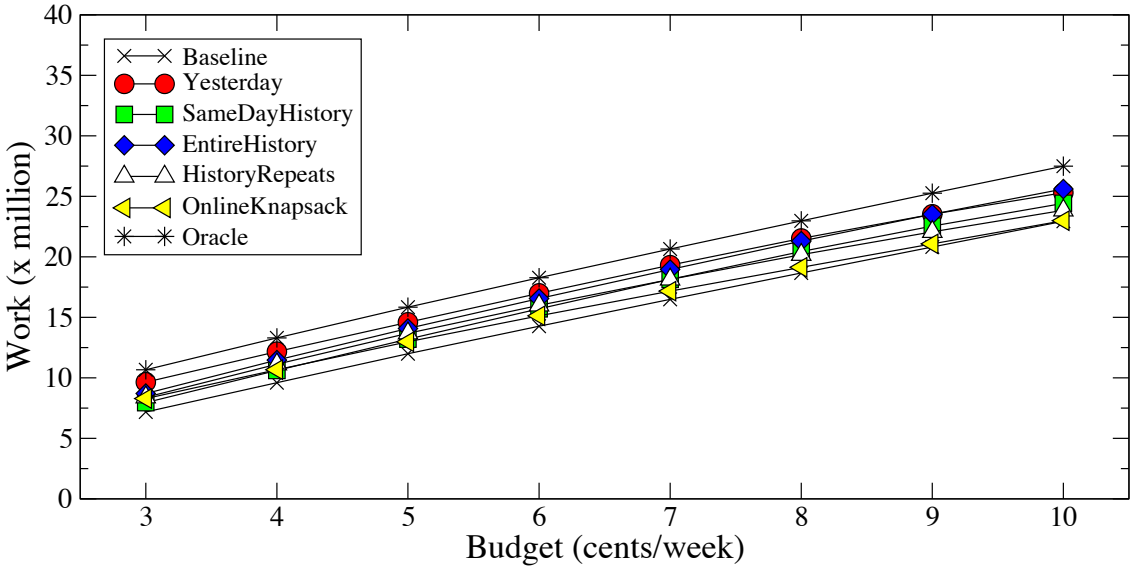

Figure 3.4: Computational work distribution with respect to weekly budgets in Fall

(`Oracle`) is also closed.

In Table 3.2, we display the fraction of peers whose budget is consumed below a certain rate. According to the table, `HistoryRepeats` leads to the largest fraction of peers whose budgets are not fully consumed, i.e., it cannot effectively utilize the budgets. This is the main reason behind the relatively poor performance of this heuristic. `OnlineKnapsack` is observed to consume not all but most of the budgets. This is because the algorithm makes the decision of allocating a time slot by taking into account the fraction of budget that is already spent and allocates the time slots more aggressively when there is enough remaining budget. Nevertheless, it still does not fully utilize the budgets, especially in the spring season. The remaining heuristics follow a similar pattern and almost fully utilize the budgets, resulting in better performance. The budget utilization does not differ much between the spring and fall seasons. Both `Baseline` and `Oracle` fully utilize the available budgets.

In Fig. 3.5 and Fig. 3.6, we demonstrate the trade-off between the total amount of work done and the total electricity bill of the peers for the spring and fall seasons, respectively. As expected, `Baseline` leads to the lowest amount of work and

Table 3.2: Percentage of peers whose budget consumption is below a certain rate

	Budget consumption							
	Spring				Fall			
Heuristics	<60%	<70%	<80%	<90%	<60%	<70%	<80%	<90%
`Baseline`	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
`Yesterday`	0.00	0.00	0.04	0.18	0.00	0.00	0.02	0.13
`SameDayHistory`	0.00	0.07	0.41	1.10	0.00	0.00	0.01	0.09
`EntireHistory`	0.00	0.16	0.60	1.58	0.00	0.00	0.00	0.01
`HistoryRepeats`	0.00	0.01	0.60	8.03	0.00	0.24	2.07	11.41
`OnlineKnapsack`	0.00	0.00	0.11	7.99	0.00	0.00	0.00	2.06
`Oracle`	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

the highest electricity bill since it is completely blind to the variation in electricity prices and fully consumes the budget of each peer. In the fall season, the `Yesterday`, `EntireHistory`, and `HistoryRepeats` heuristics yield the largest amount of work while `HistoryRepeats` achieves a similar performance with a much lower electricity bill. This also holds for `OnlineKnapsack`. Although `Oracle` attains the best performance in terms of the amount of work done, it has a rather poor performance in reducing the total electricity bill since this is not the main optimization objective in our problem.

Fig. 3.7 presents the computational work distribution with respect to countries. We observe that, although the number of peers participating from Turkey is less than those from Germany and the UK (see the discussion in Section 3.3.2), the computational work done by the peers in Turkey is much more than the work done by the peers in Germany and the UK. This is a consequence of the electricity price differences between these countries: Germany and the UK are the two countries having the most expensive electricity prices and Turkey has cheaper electricity compared to them. Consequently, the dispatcher is likely to allocate more time slots from the peers in Turkey. The same outcome is observed when Russia and Germany are compared.

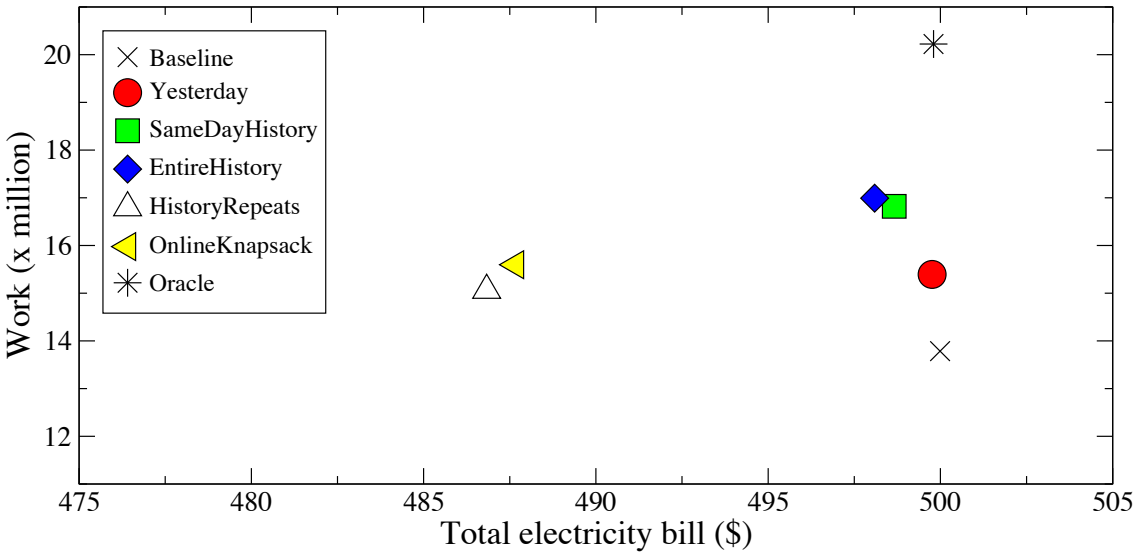

Figure 3.5: The total electricity bill of the peers versus the amount of work done in Spring

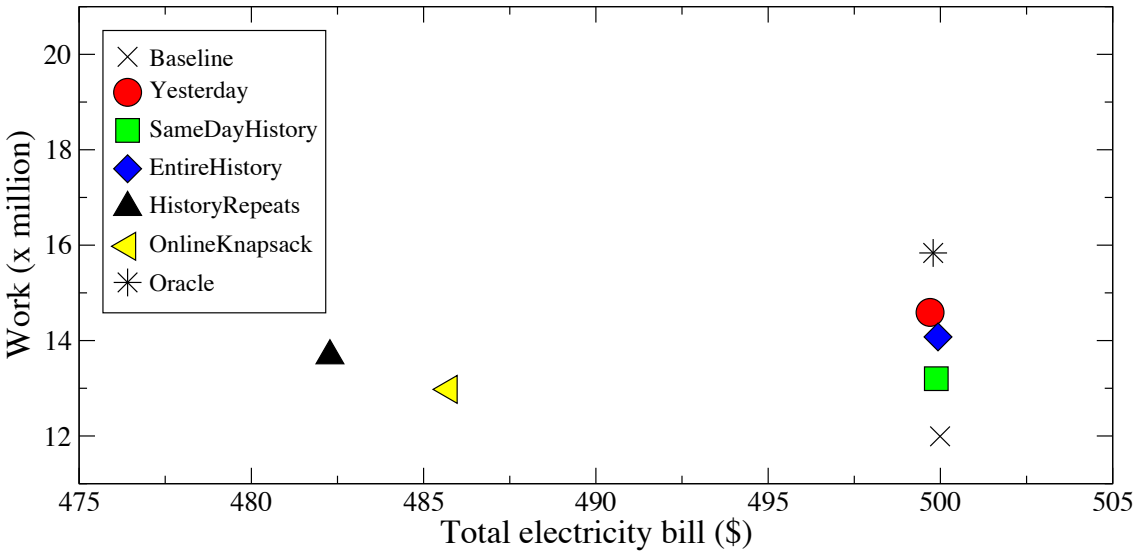

Figure 3.6: The total electricity bill of the peers versus the amount of work done in Fall

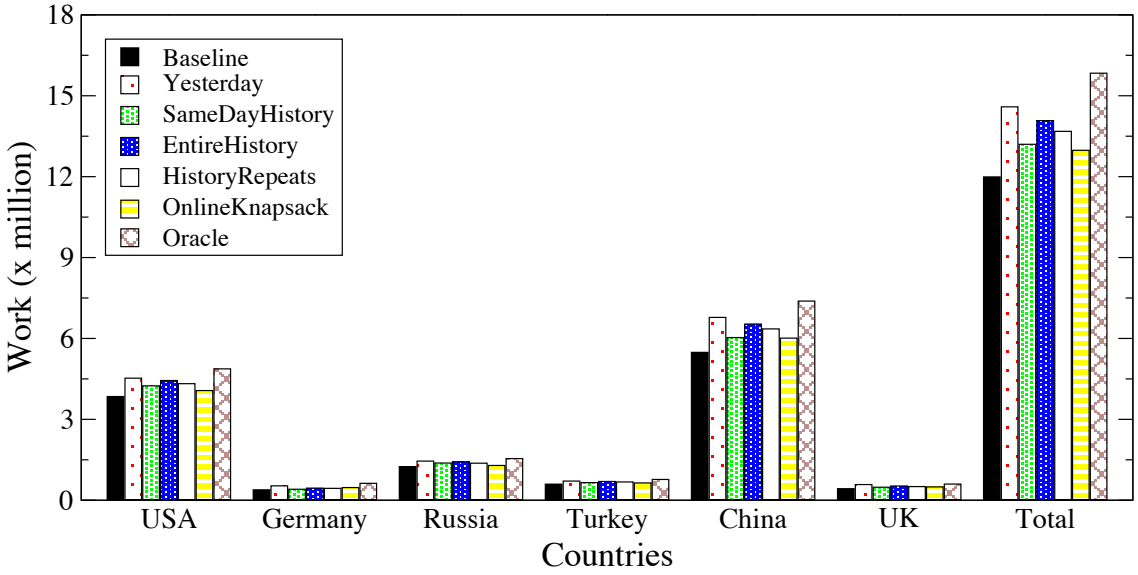

Figure 3.7: Computational work distribution with respect to countries.

Even though Germany has two orders of magnitude more peers than Russia, the total computational work done by the peers from Russia is three times larger than the work done by the peers from Germany. It is also important to note that, although the `OnlineKnapsack` algorithm performs poorly, it is the best performing heuristic in Germany, where the electricity price is the most expensive.

Fig. 3.8 shows the dissection of the total amount of work among the days of the week. We observe that most time slots are allocated on Monday. `Baseline` is observed to consume all of the available budget before Thursday. The heuristics relying on the expected electricity price try to allocate tasks in early days, but are unable to extend the task allocation to Sunday, which has a relatively low average electricity price (see Fig. 3.1). This is the main reason for these heuristics to fall behind `Oracle`, which allocates most tasks on Sunday. `HistoryRepeats` is observed to allocate a relatively large fraction of the time slots on Sunday. This is because the heuristic tends to skip previous days without allocating many tasks. Since the number of peers available on Sunday is not sufficient enough for the heuristic to allocate all remaining tasks, it fails to fully utilize the budgets. We had already pointed to this under-utilization issue

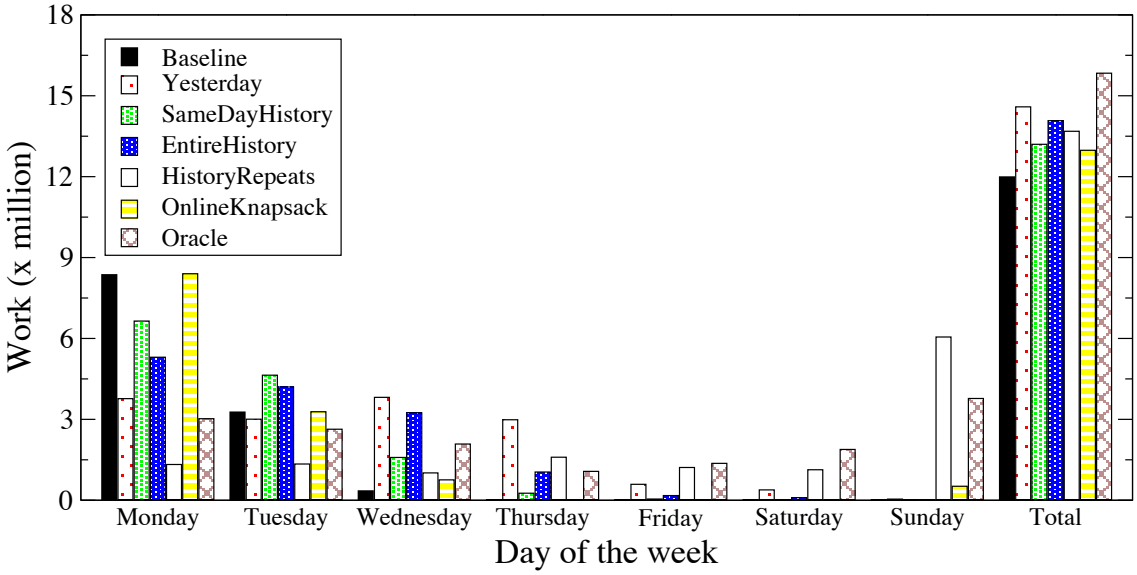

Figure 3.8: Computational work distribution with respect to the day of the week.

while discussing the results in Table 3.2.

In Fig. 3.9, we display the hourly computational work distribution. This analysis gives more insight about the performance differences between the heuristics than the previous analysis. The two well performing heuristics (`Yesterday` and `EntireHistory`) mostly exploit the resources at night time between 22:00 and 04:00 when the electricity prices are relatively lower. However, only a limited number of slots can be allocated during the night since there are relatively fewer online peers. `HistoryRepeats` performs closer to these two heuristics. As discussed before, however, it needs more effort to fully utilize peer budgets to get closer to `Oracle`. `SameDayHistory` seems to fall behind the other heuristics since it does not effectively utilize night times. `Baseline` and `OnlineKnapsack` are observed to allocate the time slots in a uniform manner in each hour, i.e., they are unable to distinguish cheaper time slots. That is the reason why these heuristics perform worse than the other heuristics.

Although the main objective in our problem is to maximize the total computational work done by the peers, we observed that, as a by-product, some heuris-

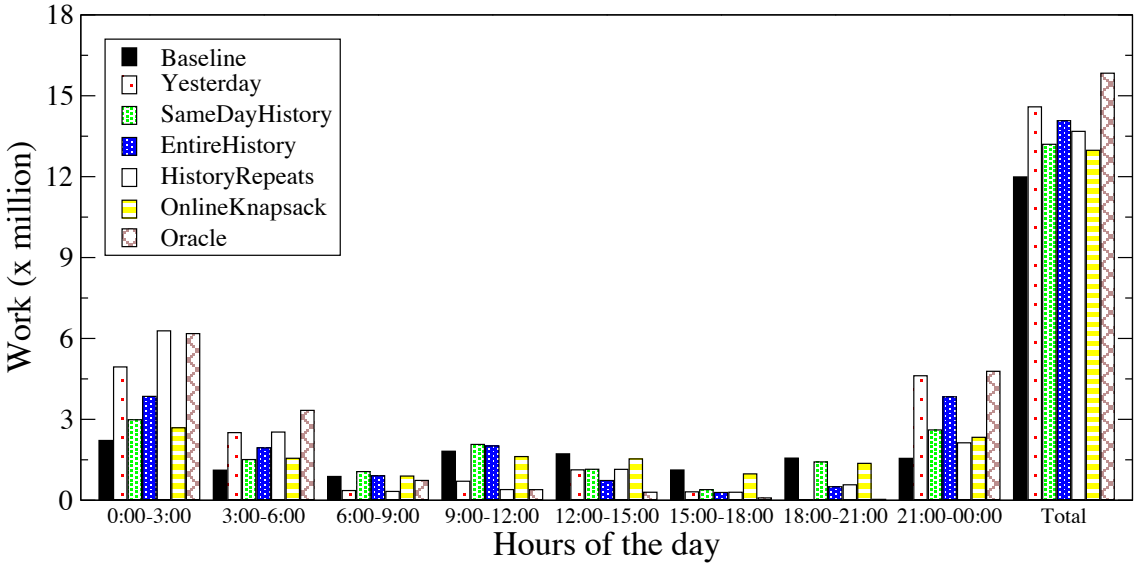

Figure 3.9: Computational work distribution with respect to the hours of the day.

tics also decreased the electricity bill of the peers relative to `Baseline`. For example, when the mean of the peer budgets is set to five cents in the fall season, the `HistoryRepeats` heuristic led to $1,770 decrease in the weekly electricity bill of the peers compared to `Baseline`. Assuming a network involving one million participants (e.g., SETI@Home), this implies that we can achieve $1,770 \times 100 = \$177,000$ saving per week. Projecting this to a whole year, the saving becomes $177,000 \times 52 = \$9,204,000$ per year.

Chapter 4

ENERGY COST AWARE JOB SCHEDULING IN GEOGRAPHICALLY DISTRIBUTED CLOUD DATA CENTERS

In this chapter, we formally state the investigated request dispatching problem as a linear programming problem and propose polynomial time heuristic solutions. Then, we present our simulation setup and provide results of our conducted simulations with performance analysis of proposed algorithms.

4.1 Problem Specification

In this section, we formally state our linear optimization problem. Table 4.1 lists the parameters and system variables used in the formulation.

Given N geographically distributed data centers, our goal is to minimize the total cost of a cloud service provider which has J jobs to run during the given time period. Service provider also need to respect QoS requirements stated in SLAs and agreed to pay penalty if any violation occurs. The total cost of the distributed data centers is defined as:

$$\sum_{i=1}^{N} E_i^{\text{total}} + Pen_i \tag{4.1}$$

where E_i^{total} is the total operational cost of data center i and Pen_i is the total penalty paid in data center i. We define total operational cost as:

$$E_i^{\text{total}} = E_i^{\text{IT}} \times PUE(T_i) \tag{4.2}$$

We introduce economization techniques that make use of cool and dry outside

weather to reduce cooling related cost. We utilize the formula given in [Xu et al., 2013] that calculates the power usage effectiveness (PUE) values as a function of outside temperature. The PUE of DC i with outside temperature T_i is defined as:

$$PUE(T_i) = 7.1705 \times 10^{-}5 \times T_i^2 + 0.0041 \times T_i + 1.0743 \tag{4.3}$$

E_i^{IT} indicates the total IT cost of data center i and is calculated as:

$$E_i^{\mathrm{IT}} = \sum_{s \in S_i} \sum_{t=0}^{\lfloor T/u \rfloor} P_{s,t} \times E_i(t) \times u \tag{4.4}$$

where $E_i(t)$ is the unit electricity cost at data center i and $P_{s,t}$ is the power consumption of server s in time slot $[t, t+u)$. We calculate $P_{s,t}$ in terms of idle power usage of a server, P_s^{idle}, and power usage of that server at peak, P_s^{peak}. We make use of the equation below reported in [Ren et al., 2012] to calculate that power usage.

$$P_{s,t} = P_s^{\mathrm{idle}} + (P_s^{\mathrm{peak}} - P_s^{\mathrm{idle}}) \times x_{s,t} \tag{4.5}$$

and $x_{s,t}$ is a decision variable to indicate whether server s is busy operating in time slot $[t, t+u)$ that will help us to determine power usage of that server:

$$x_{s,t} = \begin{cases} 1, & \text{if server } s \text{ is busy operating} \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

Since SLAs between service providers and customers are classified documents, we introduce a penalty rate parameter to fine providers if they are unable to serve a customer's job within its agreed deadline. Normally, penalty fee defined over revenue agreed in SLA. However, in our system we do not include revenues, but we assume that revenue of a certain job is proportional to the cost of running that job, thus we apply penalty fee over cost of running that particular job instance. Later in Sec. 3.3, we define different penalty rates, p_{rate}, and policies and observe their respective outcomes. Total penalty paid in a data center i is defined as the penalty sum of all delayed jobs in that data center:

$$Pen_i = \sum_{j=1}^{J} \sum_{s \in S_i} \sum_{t=0}^{\lfloor T/u \rfloor} Pen_{j,s,t} \tag{4.7}$$

where $Pen_{j,s,t}$ is the penalty cost of job j which is scheduled to run on server s in time slot $[t, t+u)$ and it is calculated as:

$$Pen_{j,s,t} = \sum_{t=Ts_j}^{Ts_j+T_{j,s}} (P_{s,t} \times E_i(t) \times u \times r_{j,s}) \times p_{rate} \tag{4.8}$$

where $r_{j,s}$ is a decision variable indicating whether job j is dispatched to server s and its deadline, Td_j, is missed:

$$r_{j,s} = \begin{cases} 1, & \text{if } Ts_j + T_{j,s} > Td_j \text{ and } j \text{ is assigned to } s \\ 0, & \text{otherwise.} \end{cases} \tag{4.9}$$

and Ts_j is the submission time of job j and $E_i(t)$ is the unit electricity price in data center i. In addition to that, data centers may have servers with different configurations (i.e. heterogeneous), thus time of running a job j on a server s, $T_{j,s}$, may differ depending on number of CPUs (c_j) and CPU frequency (f_j) that job j requires and length of that job (T_j) as well as server configurations including number of CPUs (c_s) and CPU frequency (f_s) of server s. Thus, $T_{j,s}$ is defined by:

$$T_{j,s} = \frac{c_j \times f_j \times T_j}{c_s \times f_s}, \tag{4.10}$$

4.2 Proposed Scheduling Algorithms

We assume that there exists a central scheduler that works as a proxy and receives incoming job requests. Then, this central entity schedules each job to one of the idle server located in geographically distributed data centers for processing. In such a setting, keeping our objective function in mind, we have three important factors that we can exploit:

- First is the spatial electricity price variation observed in different locations around the world.

- Second, temporal electricity price variation observed within a day or, in general sense, in time.

- Third is the opportunity to reduce cooling cost by utilizing economization systems in cooler climates (free cooling).

Considering these three factors, we propose two types of request dispatching algorithms. In the first type, each incoming job request is immediately scheduled to an available server. The algorithms of the second type can schedule jobs ahead of time if they can forecast that the electricity price will be lower in the future.

4.2.1 Immediate Scheduling Algorithms

The jobs are immediately scheduled in FCFS order.

Random dispatch (`Random`): This is a naive baseline algorithm we used for comparing the performance of our proposed scheduling algorithms. The benefit of using a random dispatcher in the scheduling phase is to balance the workload in each data center since it distributes jobs in a uniform manner. However, since this algorithm is oblivious to any cost related parameter, it is expected to have bad performance in terms of financial cost.

Cheapest first (`CheapestF`): As the name implies the cheapest server in terms of energy cost will be selected to run the current job in the queue. It can be considered as a simple yet effective greedy heuristic and similar heuristics were proposed before in the literature [Mazzucco and Dyachuk, 2012b, Mazzucco and Dyachuk, 2012a]. These works were aimed to select a random server from the cheapest data center in terms of electricity price. Different from early proposed solutions, our cheapest server algorithm also exploits cooling efficiency of each data center according to outside weather temperature associated with the data centers. The assumption is that the total cost

of running a job in a server in cooler locations can be cheaper even if that server is not located in the electrically cheapest data center. In other words, the algorithm runs the job on the server with the lowest expected total cost. If the server with the lowest total cost is busy operating another job then the second cheapest server is selected. The procedure continues until the job is scheduled. If all servers are busy operating at that time, then the job is put in the queue again to be scheduled in the next time slot.

This algorithm exploits spatial variations in the electricity price and possible reduced cooling opportunities, yet it lacks to utilize temporal variations in the electricity price.

4.2.2 Delayed Scheduling Algorithm (`LookAhead`*)*

`CheapestF` aims at scheduling the jobs in the current time slot with the least cost. However, it is possible to postpone the execution of a job to future time slots if we can somehow identify that the current electricity is expensive. To this end, we make use of historical electricity prices to determine whether to schedule the job immediately or delay its execution to a later time slot. As in the `CheapestF` algorithm, we examine each server for all time slots (i.e. current and future time slots) and select the best server and time slot combination in terms of the total electricity cost. In this approach, we exploit all three opportunities listed above, in particular, variations in spatial and temporal electricity price and reduced cooling cost of servers that are located in colder climates. Since we have the option to schedule some jobs to future time slots, there is the possibility to violate deadline constraints as stated in SLAs. Our algorithm considers any possible SLA violation in such cases and include that penalty to the expected operational cost.

In addition to using a time buffer while deciding expected energy cost, we also considered an internal job ordering strategy in this algorithm to observed its effects on the overall performance. Apart from first come first serve (FCFS), these ordering are longest job first (LJF), shortest job first (SJF) and earliest deadline first (EDF).

Algorithm 3 `LookAhead` Algorithm.

$totalCost \leftarrow 0$
$t \leftarrow 0$
while $t < T$ **do**
  $updateJobQueue()$
  for all j in the job queue **do**
    while $i < N$ **do**
      for all s in data center i **do**
        for all $t < T$ **do**
          find and save $estimateCost(j, s, t)$
      select the cheapest s, t for j
      assign j to s at time t
while $i < N$ **do**
  $E_i^{\text{total}} \leftarrow calculateCost(i)$
  $totalCost \mathrel{+}= E_i^{\text{total}} + Pen_i$
return $totalCost$

However, we did not observe significant performance improvements in our simulations and we only reported results of earliest deadline first ordering in Sec. 4.4. We illustrate the above-mentioned procedure in Algorithm 3.

4.3 Simulation Setup

We simulated a geographically distributed data center network to evaluate the performance of the proposed algorithms. The simulator and the algorithms are implemented in Java. We only consider delay-tolerant batch jobs and use the Grid5000 logs for incoming job requests.[1] The job requests contain job specific information including submission time, run time, required number of CPU cores, and other related information. We run our simulation for a week and time slots are set to one hour, so

[1]Grid5000, `http://gwa.ewi.tudelft.nl/pmwiki/pmwiki.php?n=Workloads.Gwa-t-2`.

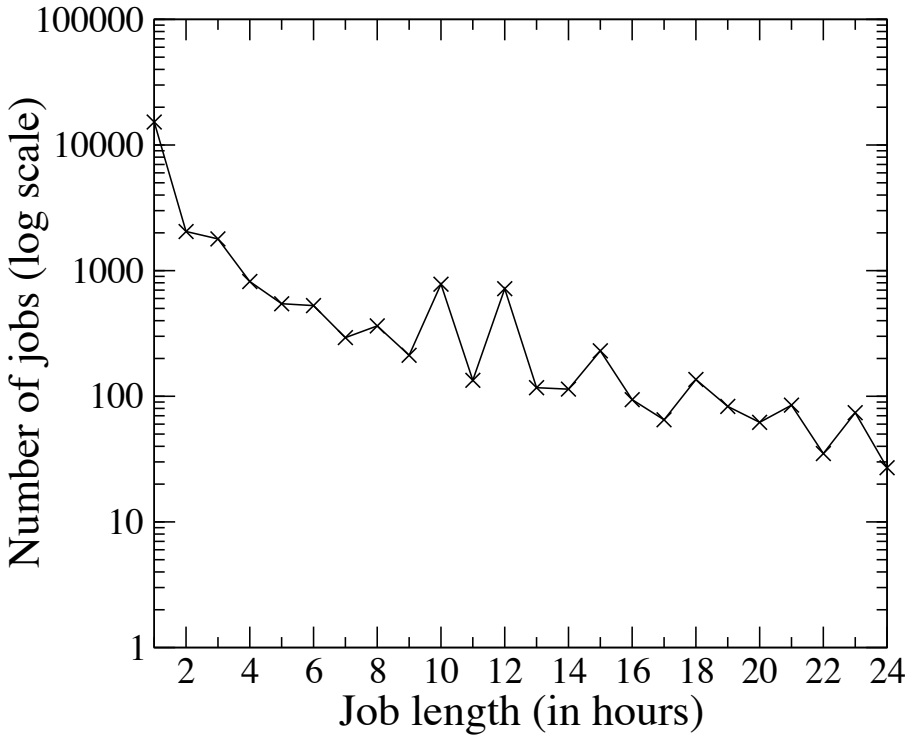

Figure 4.1: Distribution of jobs according to length.

we filtered the request logs of Grid5000 in a way that run time of any job cannot exceed 24 hours. Job length distribution of our log is illustrated in Fig. 4.1 and we also illustrate job arrival times in Fig. 4.2. However, it is important to note that our proposed algorithms are oblivious to these job distributions and make no assumptions about them. Results of `Random` algorithm is average of 25 runs.

We simulated six homogeneous data centers preferably places where Google is known to have their data centers, in particular, San Diego, California; Chicago, Illinois; Santiago, Chile; Helsinki, Finland; Dublin, Ireland and Singapore, Singapore. Our problem formulation is generalizable to heterogeneous data centers, however, in our simulations we prefer to have homogeneous data centers in terms of hardware since those big companies like Google mostly utilizes homogeneous hardware farms [Barroso and Hölzle, 2009]. Each data center is given 100 servers with Xeon architecture and four core CPUs running at 2.66 GHZ [Goiri et al., 2012]. We used

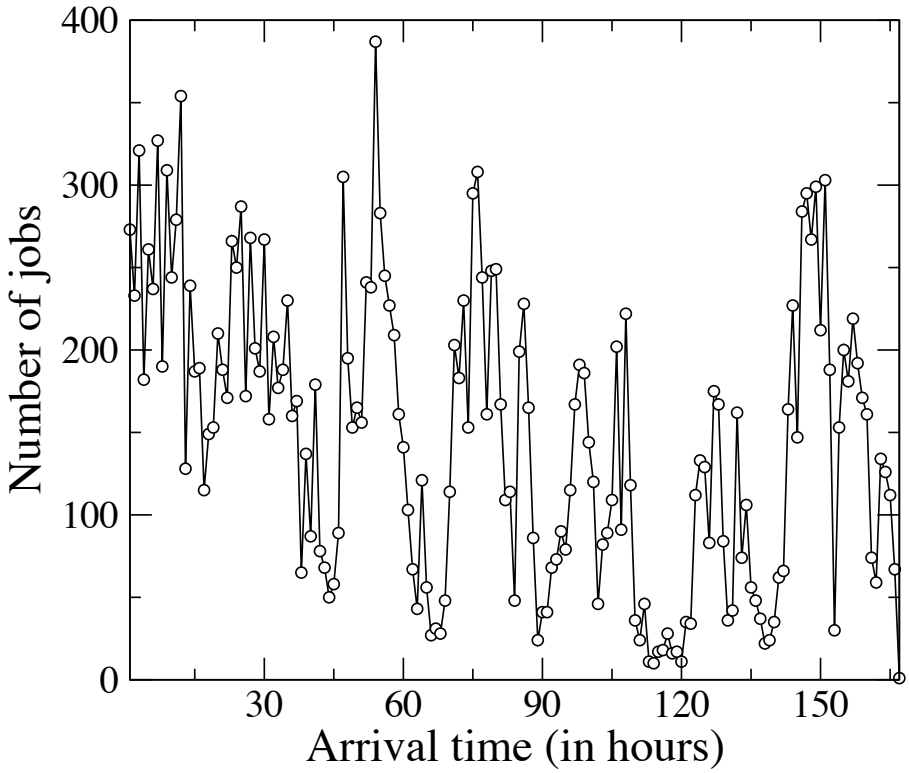

Figure 4.2: Distribution of jobs according to arrival time.

real electricity price traces for San Diego and Chicago from FERC (Federal Energy Regulatory Commission of the USA),[2] and scaled the prices for other countries according to the country-wide average prices.[3] Average temperatures values are gathered from Wunderground [4] and these temperatures are given in Table. 4.2 along with the PUE values calculated by Eq. 4.3 and these PUE values are consistent with the ones stated in [Barroso and Hölzle, 2009].

While considering any SLA related penalty, we introduce three policies to determine how much should be added to the cost. These policies are:

- fix penalty, *Fixed*.

[2]FERC: Electric Power Markets, `http://www.ferc.gov/market-oversight/mkt-electric/overview.asp`.

[3]Wikipedia-Electricity Pricing, `http://en.wikipedia.org/wiki/Electricity_pricing`.

[4]Wunderground, `http://www.wunderground.com/history`.

- penalty based on job length, *Length*.
- penalty based on job length and delay of the delivery of the job, *Length&Delay*.

We introduce a penalty rate, p_{rate}, for each of these policies. Since we do not utilize any revenue in our problem formulation, we set p_{rate} to be some fraction of the cost of running that job. In other words, we calculate penalty over the cost and add that amount to the total cost later. In order to test the adaptability of our system, we run our simulations with different p_{rate}'s, in particular 1%, 5% and 10%. In fix penalty scheme, we calculate an average cost of running a single job and apply p_{rate} over that cost. In job length based penalty scheme, we apply penalty over cost of running that particular job and job length is the key factor in calculating that cost. Finally, in penalty scheme based on job length and lateness of the delivery of the job, we fix p_{rate} to 1% and multiply that rate with the number of time slots the deadline is missed to find the final rate.

4.4 Experimental Results

As explained in Section 4.2, there are three factors that we consider: spatial electricity price change, temporal electricity price change and outside weather temperature. By comparing the performance of the algorithms, we can determine the contribution of these factors to the cost saving.

Table. 4.3 summarizes our findings and the amount of improvement achieved by each algorithm when deadlines are strict and cannot be violated. Since deadlines cannot be postponed, we sometimes end up with unscheduled jobs in `LookAhead` algorithm because some of the early time slots are not utilized by this approach if it forecasts some cheaper future time slots especially during peak times in job requests. In the end, we were left with some non-executed jobs in the job queue and that creates a big obstacle in user satisfaction and cannot be explicitly reflected to total cost. Regarding that, we then introduce penalty clauses as it is the case in real applications via SLAs. However, it is helpful to analyze the results in such settings to observe the

incremental effects of the three factors mentioned above. We test the performance of the algorithms with different deadlines selected uniformly from different intervals, in particular 4, 16 and 64 hours ahead of their submission time plus run time. As expected performance of our `LookAhead` algorithm improves with increasing deadline since we have more flexibility to postpone the execution of jobs. In particular its performance improves from 3.5% (when deadline interval is 1-4 hours) to 5.5% (when deadline interval is 1-64 hours) compared to `Random`. Most of the gain is achieved by exploiting the temporal electricity price variation as seen by the performance improvement between the `CheapestF` and `LookAhead` since the only difference between these algorithms is that `LookAhead` also exploits the temporal variations in prices by scheduling jobs to future time slots and it improves over `CheapestF` by 3% and improves over `Random` by 5.5%. Next, the biggest improvement is achieved by exploiting spatial electricity price and temperature variations as observed by the improvement by `CheapestF` over `Random` which is around 2.5%.

Table 4.1: System parameters

Notation	Description
c_j	Number of CPUs job j requires
c_s	Number of CPU cores server s has
E_i^{IT}	Total IT cost of servers in DC i (\$)
E_i^{total}	Total energy cost of DC i (\$)
$E_i(t)$	Electricity price of DC i in time slot $[t, t+u)$
f_j	CPU frequency job j requires
f_s	CPU frequency of server s
J	Number of jobs
N	Number of Data Center (DC)
p_{rate}	Penalty rate
$Pen_{j,s,t}$	Penalty of job j on server s in time slot $[t, t+u)$
Pen_i	Total penalty paid in DC i
P_s^{idle}	Power consumption of server s when idle (Watt/u)
P_s^{peak}	Power consumption of server s at peak (Watt/u)
$P_{s,t}$	Power consumption of server s in time slot $[t, t+u)$
S_i	Set of servers in DC i
Ts_j	Submission time of job j
T_j	Total number of time slots job j requires (job length)
Td_j	Deadline of job j
$T_{j,s}$	Number of time slots required to process job j in server s
u	Length of a unit time slot

Table 4.2: PUE of Simulated Data Centers

Location	Temperature	PUE
San Diego	15 °C	1.15
Chicago	1 °C	1.07
Santiago	22 °C	1.19
Helsinki	−9 °C	1.04
Dublin	7 °C	1.10
Singapore	28 °C	1.24

Table 4.3: Performance of different heuristics (no penalty).

		Incomp.	Total	Improvement over (%)	
Deadline	Heuristics	jobs	cost ($)	`Random`	`CheapestF`
	`Random`	0	3,904.03	–	–
4	`CheapestF`	0	3,817.17	2.20	–
	`LookAhead`	31	3,752.76	3.87	1.69
	`Random`	0	3,904.03	–	–
16	`CheapestF`	0	3,817.17	2.25	–
	`LookAhead`	10	3,735.07	4.33	2.15
	`Random`	0	3,904.03	–	–
64	`CheapestF`	0	3,817.17	2.24	–
	`LookAhead`	0	3,696.33	5.32	3.17

Table 4.4: Performance of the `LookAhead` heuristic (idle servers are not switched off).

Penalty type	Deadline	Penalty rate	# of jobs with penalty	Cost ($)			Improvement over (%)	
				IT & cooling	Penalty	Total	`Random`	`CheapestF`
Fixed	4	0.01	7,966	3,725.43	12.58	3,738.01	4.25	2.07
		0.05	790	3,748.76	6.24	3,755.00	3.82	1.63
		0.10	235	3,752.90	3.71	3,756.61	3.78	1.59
	16	0.01	4,781	3,717.11	7.50	3,724.61	4.60	2.42
		0.05	533	3,730.84	4.21	3,735.05	4.33	2.15
		0.10	28	3,735.23	0.44	3,735.67	4.31	2.14
	64	0.01	744	3,692.85	1.17	3,694.02	5.38	3.23
		0.05	47	3,695.57	0.37	3,695.94	5.33	3.18
		0.10	0	3,696.20	0.00	3,696.20	5.32	3.17
Length	4	0.01	18,267	3,710.19	3.72	3,713.91	4.87	2.71
		0.05	14,876	3,719.83	13.52	3,733.35	4.37	2.20
		0.10	13,316	3,728.25	20.68	3,748.93	3.97	1.79
	16	0.01	14,612	3,703.25	3.39	3,706.64	5.06	2.90
		0.05	10,291	3,713.71	10.25	3,723.96	4.61	2.44
		0.10	7,694	3,721.01	13.47	3,734.48	4.34	2.17
	64	0.01	5,050	3,690.59	1.15	3,691.74	5.44	3.29
		0.05	1,493	3,693.05	1.81	3,694.86	5.36	3.20
		0.10	823	3,695.09	1.58	3,696.67	5.31	3.16
Length & delay	4	0.01	19,378	3,707.48	69.58	3,777.06	3.25	1.05
	16	0.01	16,442	3,703.72	50.78	3,754.50	3.83	1.64
	64	0.01	6,966	3,689.58	9.59	3,699.17	5.25	3.09

In order to overcome the non-executed jobs problem, we add penalty clauses and increase the total cost if any of the deadlines are missed. As explained in Sec. 3.3 in detail, we tested different p_{rate} scenarios and penalty policies. Since `Random` and `CheapestF` immediately schedule all of the incoming jobs, they have no penalized jobs and changing penalty policies or penalty rates have no effect on their performance. However, they are introduced to validate the stability of our proposed `LookAhead` heuristic against different SLAs. In Table. 4.4 we illustrate performance of our algorithms with different penalty policies, changing p_{rate}s and deadline intervals. Improvements are close to what we observed in no penalty case, however, now we are able to schedule all of the jobs in the job queue without leaving no job non-executed.

In general, effects of deadline change are similar in all penalty policies. When the deadline interval increases, we have more flexibility to schedule jobs to future cheaper time slots without paying any penalty for the delay. Similarly, sometimes we may even use penalty clauses in our favor and prefer to pay penalty if, in total, it will be cheaper to run that job later on. For example, if we set fix penalty policy with p_{rate} of 0.01 when we increase deadline interval from 4 hours to 16 hours and then 64 hours, improvement of `LookAhead` algorithm over `Random` is increasing from 4% to 4.5% and to 5.5%, respectively and from 2% to 2.5% and to 3.5% over `CheapestF`, respectively. Moreover, with the same reason explained above, number of penalized jobs is also decreasing with increasing deadline intervals since we can delay more jobs without having to pay penalty.

We also run our simulations with varying penalty rates, p_{rate}, in particular with 0.01, 0.05 and 0.1. We observed that performance of `LookAhead` algorithm degrades with increasing penalty rates, which is an expected outcome. Increasing penalty rate implies that we will be charged more if we miss the deadline of a job. Thus, our flexibility to schedule jobs ahead in time decreases with the cost of paying too much penalty. However, we are still able to improve the performance of `Random` and `CheapestF` by 4% and 2%, respectively even with strict deadlines, i.e. when p_{rate} is equal to 0.1. As expected, number of penalized jobs also decreases with increasing

penalty rates, because it will be less likely to miss the deadline of a job to avoid high penalty costs.

Finally, we define three types of penalty policies, namely; *Fixed*, *Length* and *Length&Delay*. Overall, `LookAhead` algorithm is able to perform similar in each of the penalty policy, thus it is not vulnerable to changing SLA conditions. In particular, depending on p_{rate} and deadline interval, it performs better in *Length* policy with around $4-5\%$ improvement over `Random` and $2-3\%$ over `CheapestF`. In *Fixed* policy these improvements drop around to 4% over `Random` and 2% over `CheapestF`. In *Length&Delay* policy, performance improvement of our proposed algorithm is still close to other policies and around $3-5\%$ over `Random` and $1-3\%$ over `CheapestF`.

In addition to exploiting price and temperature variations, we tested our data center network with imaginary power proportional servers, i.e. if any server stays idle in a time slot, we assume it consumes no power at all, in other words they are switched off. There are several proposed solutions in the literature with the exact same purpose, i.e. avoiding idle power consumption [Zhang et al., 2013, Mazzucco and Dyachuk, 2012b, Mazzucco and Dyachuk, 2012a]. Since idle power consumption will be fully eliminated, total energy consumption and energy cost of the overall system will be far less than the original case. There are several conditions that needs to be considered while switching off servers like the time delay of server openings and closure, yet our intention here is to observe the performance of our algorithms if power proportional servers are utilized in cloud data centers. In that regard, we do not compare our results with our findings in Table. 4.4 (no server switch off) but rather we analyze relative performance improvement of our `LookAhead` algorithm against the `CheapestF` heuristic and `Random` baseline. Table. 4.5 summarizes our findings when we switch off any idle server in any time slot. The most important observation is that we are able to almost triple the performance of `LookAhead` algorithm if server switching off feature can be applied in cloud data centers. In particular, `LookAhead` achieves to improve the `Random` and `CheapestF` by 15% and 10% respectively when we use penalty clauses based on job length with p_{rate} equal to 0.01 and deadline interval set to 1-64 hours.

On the other hand, the improvements were 5.5% and 3.5% respectively, with the very same configurations (same penalty policy, p_{rate} and deadline interval) when we did not switch off any server. Apart from that, our findings are similar to the previous case with no server switching off even if they are idle. However, these results show that even greater cost benefits are achievable if we can utilize power proportional servers and/or introduce effective server switching off techniques.

Table 4.5: Performance of the `LookAhead` heuristic (idle servers are switched off).

Penalty type	Deadline	Penalty rate	# of jobs with penalty	Cost ($)			Improvement over (%)	
				IT & cooling	Penalty	Total	`Random`	`CheapestF`
Fixed	4	0.01	7,898	2,195.19	12.47	2,207.66	12.24	6.42
		0.05	793	2,238.56	6.20	2,244.76	10.76	4.85
		0.10	214	2,243.15	3.38	2,246.53	10.69	4.77
	16	0.01	4,841	2,179.72	7.64	2,187.36	13.04	7.28
		0.05	505	2,205.18	4.40	2,209.58	12.16	6.34
		0.10	37	2,213.73	0.58	2,214.31	11.97	6.14
	64	0.01	794	2,134.41	1.25	2,135.66	15.10	9.47
		0.05	50	2,140.75	0.39	2,141.14	14.88	9.24
		0.10	0	2,141.60	0.00	2,141.60	14.86	9.22
Length	4	0.01	18,280	2,167.41	3.71	2,171.12	13.69	7.97
		0.05	14,821	2,185.62	12.57	2,198.19	12.61	6.82
		0.10	13,291	2,200.20	20.63	2,220.83	11.71	5.86
	16	0.01	14,644	2,158.62	3.20	2,161.82	14.06	8.36
		0.05	10,144	2,173.16	10.12	2,183.28	13.21	7.45
		0.10	7,461	2,187.36	13.38	2,200.74	12.51	6.71
	64	0.01	5,040	2,132.83	1.16	2,133.99	15.17	9.54
		0.05	1,436	2,136.37	1.83	2,138.20	15.00	9.36
		0.10	842	2,141.78	1.69	2,143.47	14.79	9.14
Length & delay	4	0.01	19,380	2,162.89	69.40	2,232.29	11.26	5.37
	16	0.05	16,486	2,155.87	50.78	2,206.65	12.28	6.46
	64	0.10	7,087	2,132.54	9.46	2,142.00	14.85	9.20

As a side note, in `LookAhead` algorithm, we use historical electricity price averages to determine which time slot will be the cheapest to run the job and actual unit electricity price may be different than what we assume. However, some companies prefer to purchase electricity from day-ahead markets in which they know the actual unit electricity price they will pay 24 hours in advance. Our proposed algorithm will be likely to have better performance in such cases, since we will have the full knowledge of electricity prices.

Chapter 5

CONCLUSION

Energy consumption and corresponding energy costs in large scale distributed systems is an important problem that attracted researchers from both industry and academia. In this thesis, we have investigated two different types of distributed systems, in particular, volunteer computing networks and geographically distributed cloud data centers.

In the first part, we proposed the idea of having monetary budgets in volunteer computing networks, limiting the financial burden incurred on the peers due to the usage of their computational resources by the volunteer computing network. We showed that, under the assumption of temporal volatility in electricity prices, this idea leads to an interesting task allocation problem. We formally specified this task allocation problem and proposed various heuristics as potential solutions. Simulations using real-life electricity price data demonstrated that the proposed heuristics can increase the amount of useful work done in the network while respecting the peers' budgets, compared to a simple baseline.

In particular, in the fall season, the amount of work done by our best performing heuristic, `Yesterday`, is 20% higher than the work done by `Baseline` and only 8% less than the work done by `Oracle`, on average. On the other hand, in the spring season, `SameDayHistory` achieves to improve the performance of `Baseline` by 19% and fall behind of `Oracle` by 14%. We believe that there is still some room for improvement in the performance of `HistoryRepeats` since it exhibits good performance behavior especially in hour of the day analysis. `OnlineKnapsack` manages to beat `Baseline` although it generally performs worse than the other heuristics. On the other hand, as mentioned in Section 3.4, we observe this algorithm to perform better in countries

where the electricity price is high. In general, this may imply that we can select and use different heuristics depending on the country.

Some of the ideas in this work can also be applied to P2P clouds, which have been recently envisioned as a promising computing model [Babaoglu et al., 2011, Panzieri et al., 2011]. In contrast to the centralized and federated cloud architectures, a P2P cloud operates in a fully distributed manner providing on-demand scalability. A P2P cloud system allows building a computing infrastructure with independent resources that can be assembled in a distributed manner to serve different tasks. This model brings new business opportunities. Example applications that can be hosted on a P2P cloud are loosely coupled distributed applications for which the location of peers is important to provide resources, data, and computation close to the client [Babaoglu et al., 2011]. Furthermore, computation-intensive parallel applications and communication-intensive data/video delivery systems can benefit from a P2P cloud.[1] Important issues in a P2P cloud setting are efficient resource partitioning and allocation among several tasks, as well as efficient management of tasks. Although there exist recent studies on P2P cloud systems [Babaoglu et al., 2011, Babaoglu et al., 2006, Cunsolo et al., 2009a, Cunsolo et al., 2009b, Valancius et al., 2009], no work has considered the energy price aspect in allocating resources and managing tasks.

In the second part, we investigated the problem of massive energy cost of geographically distributed cloud data centers. We first mathematically formulated the optimization problem as a linear programming problem by including energy cost of servers and cooling related energy costs. We also utilized SLAs to ensure user satisfaction and introduced penalty clauses if any of the deadline constraints are not satisfied. We proposed two workload scheduling algorithms that run in polynomial time and exploit spatio-temporal electricity price variations and free cooling opportunities in colder climates to minimize the energy cost of service providers. Our extensive sim-

[1] Clouds and Peer-to-Peer, `http://berkeleyclouds.blogspot.com/2009/06/clouds-and-peer-to-peer.html`.

ulations with real world electricity prices and workload data show that significant electricity cost reductions can be achieved by the proposed algorithms, compared to a random scheduler that aims to achieve only load balancing between data centers. In particular, we are able to improve the total cost up to 6% with `LookAhead` algorithm which corresponds to cost savings of millions of dollars when we think of large industrial companies. We are also planning to integrate different cost and energy saving techniques into our system including virtual machine migration and switching off idle servers as a future work.

BIBLIOGRAPHY

[Anderson, 2011] Anderson, D. (2011). Emulating volunteer computing scheduling policies. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, pages 1839–1846.

[Anderson, 2007] Anderson, D. P. (2007). Local scheduling for volunteer computing. *Proceedings of the 2007 IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8.

[Babaioff et al., 2007] Babaioff, M., Immorlica, N., Kempe, D., and Kleinberg, R. (2007). A knapsack secretary problem with applications. In *Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization*, pages 16–28.

[Babaoglu et al., 2006] Babaoglu, O., Jelasity, M., Kermarrec, A., Montresor, A., and van Steen, M. (2006). Managing clouds: a case for a fresh look at large unreliable dynamic networks. In *Operating Systems Review*, pages 9–13.

[Babaoglu et al., 2011] Babaoglu, O., Marzolla, M., and Tamburini, M. (2011). Design and implementation of a P2P cloud system. Technical Report UBLCS-2011-10, Department of Computer Science, University of Bologna.

[Barroso and Hölzle, 2009] Barroso, L. A. and Hölzle, U. (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 1st edition.

[Bhagwan et al., 2003] Bhagwan, R., Savage, S., and Voelker, G. M. (2003). Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, pages 256–267.

[Böckenhauer et al., 2012] Böckenhauer, H.-J., Komm, D., Královič, R., and Rossmanith, P. (2012). On the advice complexity of the knapsack problem. In *Proceedings of the 10th Latin American international conference on Theoretical Informatics*, pages 61–72.

[Buchbinder et al., 2011] Buchbinder, N., Jain, N., and Menache, I. (2011). Online job-migration for reducing the electricity bill in the cloud. In *Proceedings of the 10th International IFIP TC 6 Conference on Networking - Volume Part I*, pages 172–185.

[Chase et al., 2001] Chase, J. S., Anderson, D. C., Thakar, P. N., Vahdat, A. M., and Doyle, R. P. (2001). Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116.

[Costa et al., 2011] Costa, F., Silva, L., and Dahlin, M. (2011). Volunteer cloud computing: MapReduce over the Internet. In *IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pages 1855–1862.

[Cunsolo et al., 2009a] Cunsolo, V., Distefano, S., Puliafito, A., and Scarpa, M. (2009a). Cloud@home: Bridging the gap between volunteer and cloud computing. In *Emerging Intelligent Computing Technology and Applications*, pages 423–432.

[Cunsolo et al., 2009b] Cunsolo, V. D., Distefano, S., Puliafito, A., and Scarpa, M. (2009b). Volunteer computing and desktop cloud: The cloud@home paradigm. In *Proceedings of the 8th IEEE International Symposium on Network Computing and Applications*, pages 134–139.

[Estrada et al., 2006] Estrada, T., Flores, D. A., Taufer, M., Teller, P. J., Kerstens, A., and Anderson, D. P. (2006). The effectiveness of threshold-based scheduling policies in BOINC projects. In *Proceedings of the 2nd IEEE International Conference on e-Science and Grid Computing*, page 88.

[Gao et al., 2012] Gao, P. X., Curtis, A. R., Wong, B., and Keshav, S. (2012). It's not easy being green. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 211–222.

[Garg et al., 2011a] Garg, S., Gopalaiyengar, S., and Buyya, R. (2011a). Sla-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In *Algorithms and Architectures for Parallel Processing*, volume 7016 of *Lecture Notes in Computer Science*, pages 371–384.

[Garg et al., 2011b] Garg, S. K., Yeo, C. S., Anandasivam, A., and Buyya, R. (2011b). Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6):732–749.

[Goiri et al., 2012] Goiri, I., Berral, J. L., Fitó, J. O., Julií, F., Nou, R., Guitart, J., Gavaldí, R., and Torres, J. (2012). Energy-efficient and multifaceted resource management for profit-driven virtualized data centers. *Future Gener. Comput. Syst.*, 28(5):718–731.

[Guo et al., 2011] Guo, Y., Ding, Z., Fang, Y., and Wu, D. (2011). Cutting down electricity cost in internet data centers by using energy storage. In *Global Telecommunications Conference*, pages 1–5.

[He et al., 2012] He, J., Deng, X., Wu, D., Wen, Y., and Wu, D. (2012). Socially-responsible load scheduling algorithms for sustainable data centers over smart grid. In *IEEE Third International Conference on Smart Grid Communications*, pages 406–411.

[Himyr et al., 2012] Himyr, N., Blomer, J., Buncic, P., Giovannozzi, M., Gonzalez, A., Harutyunyan, A., Jones, P. L., Karneyeu, A., Marquina, M. A., Mcintosh, E., Segal, B., Skands, P., Grey, F., Gonzlez, D. L., and Zacharov, I. (2012). BOINC

service for volunteer cloud computing. *Journal of Physics: Conference Series*, 396(3).

[Kayaaslan et al., 2011] Kayaaslan, E., Cambazoglu, B. B., Blanco, R., Junqueira, F. P., and Aykanat, C. (2011). Energy-price-driven query processing in multi-center web search engines. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 983–992.

[Kondo et al., 2007] Kondo, D., Anderson, D. P., and Vii, J. M. (2007). Performance evaluation of scheduling policies for volunteer computing. In *Proceedings of the 3rd IEEE International Conference on e-Science and Grid Computing*, pages 415–422.

[Kondo et al., 2009] Kondo, D., Javadi, B., Malecot, P., Cappello, F., and Anderson, D. (2009). Cost-benefit analysis of cloud computing versus desktop grids. In *IEEE International Symposium on Parallel Distributed Processing*, pages 1–12.

[Krioukov et al., 2010] Krioukov, A., Mohan, P., Alspaugh, S., Keys, L., Culler, D., and Katz, R. H. (2010). Napsac: design and implementation of a power-proportional web cluster. In *Proceedings of the first ACM SIGCOMM workshop on Green networking*, pages 15–22.

[Le et al., 2009] Le, K., Bianchini, R., Martonosi, M., and Nguyen, T. D. (2009). Cost- and energy-aware load distribution across data centers. In *Workshop on Power Aware Computing and Systems*.

[Le et al., 2010] Le, K., Bilgir, O., Bianchini, R., Martonosi, M., and Nguyen, T. D. (2010). Managing the cost, energy consumption, and carbon footprint of internet services. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 357–358.

[León and Navarro, 2011] León, X. and Navarro, L. (2011). Limits of energy saving for the allocation of data center resources to networked applications. In *Proceedings*

of the 30th IEEE International Conference on Computer Communications, pages 216–220.

[Li et al., 2012] Li, J., Li, Z., Ren, K., and Liu, X. (2012). Towards optimal electric demand management for internet data centers. *IEEE Trans. Smart Grid*, 3(1):183–192.

[Lin and Ng, 2005] Lin, M. and Ng, S. M. (2005). A genetic algorithm for energy aware task scheduling in heterogeneous systems. *Parallel Processing Letters*, 15(4):439–450.

[Lin et al., 2011] Lin, M., Wierman, A., Andrew, L. L. H., and Thereska, E. (2011). Dynamic right sizing for power proportional data centers. In *INFOCOM, 2011 Proceedings IEEE*, pages 1098–1106.

[Liu et al., 2011] Liu, Z., Lin, M., Wierman, A., Low, S. H., and Andrew, L. L. (2011). Greening geographical load balancing. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 233–244.

[Lombraña González et al., 2012] Lombraña González, D., Grey, F., Blomer, J., Buncic, P., Harutyunyan, A., Marquina, M., Segal, B., Skands, P., and Karneyeu, A. (2012). Virtual machines and volunteer computing: Experience from lhc@home: Test4theory project. In *The International Symposium on Grids and Clouds*, pages 36–49.

[Mani and Rao, 2011] Mani, S. and Rao, S. (2011). Operating cost aware scheduling model for distributed servers based on global power pricing policies. In *Proceedings of the 4th Annual ACM Bangalore Conference*.

[Marchetti-Spaccamela and Vercellis, 1995] Marchetti-Spaccamela, A. and Vercellis,

C. (1995). Stochastic on-line knapsack problems. *Mathematical Programming*, pages 73–104.

[Mazzucco and Dyachuk, 2012a] Mazzucco, M. and Dyachuk, D. (2012a). Balancing electricity bill and performance in server farms with setup costs. *Future Generation Computer Systems*, 28(2):415 – 426.

[Mazzucco and Dyachuk, 2012b] Mazzucco, M. and Dyachuk, D. (2012b). Optimizing cloud providers revenues via energy efficient server allocation. *Sustainable Computing: Informatics and Systems*, 2(1):1 – 12.

[Moore et al., 2005] Moore, J., Chase, J., Ranganathan, P., and Sharma, R. (2005). Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, pages 5–5.

[Pakbaznia and Pedram, 2009] Pakbaznia, E. and Pedram, M. (2009). Minimizing data center cooling and server power costs. In *Proceedings of the 14th ACM/IEEE international symposium on Low power electronics and design*, pages 145–150.

[Panzieri et al., 2011] Panzieri, F., Babaoglu, O., Ghini, V., Ferretti, S., and Marzolla, M. (2011). Distributed computing in the 21st century: Some aspects of cloud computing. Technical Report UBLCS-2011-03, Department of Computer Science, University of Bologna.

[Qureshi et al., 2009] Qureshi, A., Weber, R., Balakrishnan, H., Guttag, J., and Maggs, B. (2009). Cutting the electric bill for Internet-scale systems. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, pages 123–134.

[Rao et al., 2010a] Rao, L., Liu, X., Ilic, M., and Liu, J. (2010a). MEC-IDC: joint load balancing and power control for distributed Internet data centers. In *Proceed-*

ings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems, pages 188–197.

[Rao et al., 2010b] Rao, L., Liu, X., Xie, L., and Liu, W. (2010b). Minimizing electricity cost: optimization of distributed Internet data centers in a multi-electricity-market environment. In *Proceedings of the 29th IEEE International Conference on Computer Communications*, pages 1145–1153.

[Ren et al., 2012] Ren, S., He, Y., and Xu, F. (2012). Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In *the 32nd International Conference on Distributed Computing Systems*.

[Sadhasivam et al., 2009] Sadhasivam, S., Nagaveni, N., Jayarani, R., and Ram, R. (2009). Design and implementation of an efficient two-level scheduler for cloud computing environment. In *International Conference on Advances in Recent Technologies in Communication and Computing*, pages 884–886.

[Sakamoto et al., 2012] Sakamoto, T., Yamada, H., Horie, H., and Kono, K. (2012). Energy price driven request dispatching for cloud data centers. In *Proceedings of the 5th IEEE International Conference on Cloud Computing*, pages 974–976.

[Sankaranarayanan et al., 2011] Sankaranarayanan, A. N., Sharangi, S., and Fedorova, A. (2011). Global cost diversity aware dispatch algorithm for heterogeneous data centers. In *Proceedings of the 2nd Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 289–294.

[Shah and Krishnan, 2008] Shah, A. J. and Krishnan, N. (2008). Optimization of global data center thermal management workload for minimal environmental and economic burden. *IEEE Transactions on Components and Packaging Technologies*, pages 39–45.

[Stanojevic and Shorten, 2010] Stanojevic, R. and Shorten, R. (2010). Distributed dynamic speed scaling. In *Proceedings of the 29th Conference on Information Communications*, pages 426–430.

[Tang et al., 2008] Tang, Q., Gupta, S. K. S., and Varsamopoulos, G. (2008). Energy-efficient thermal-aware task scheduling for homogeneous high-performance computing data centers: A cyber-physical approach. *IEEE Trans. Parallel Distrib. Syst.*, 19(11):1458–1472.

[Taufer et al., 2007] Taufer, M., Kerstens, A., Estrada, T. P., Flores, D. A., Zamudio, R., Teller, P. J., Armen, R., and Brooks, C. L. (2007). Moving volunteer computing towards knowledge-constructed, dynamically-adaptive modeling and scheduling. *Parallel and Distributed Processing Symposium*, page 478.

[Urgaonkar et al., 2011] Urgaonkar, R., Urgaonkar, B., Neely, M. J., and Sivasubramaniam, A. (2011). Optimal power cost management using stored energy in data centers. In *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, pages 221–232.

[Valancius et al., 2009] Valancius, V., Laoutaris, N., Massoulié, L., Diot, C., and Rodriguez, P. (2009). Greening the Internet with nano data centers. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, pages 37–48.

[Van den Bossche et al., 2010] Van den Bossche, R., Vanmechelen, K., and Broeckhove, J. (2010). Cost-optimal scheduling in hybrid iaas clouds for deadline constrained workloads. In *IEEE 3rd International Conference on Cloud Computing*, pages 228–235.

[Wang et al., 2013] Wang, R., Kandasamy, N., Nwankpa, C., and Kaeli, D. (2013). Datacenters as controllable load resources in the electricity market. In *The 33rd IEEE International Conference on Distributed Computing Systems*, pages 176–185.

[Wang et al., 2011] Wang, Y., Wang, X., and Zhang, Y. (2011). Leveraging thermal storage to cut the electricity bill for datacenter cooling. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, pages 8:1–8:5.

[Weerts et al., 2012] Weerts, B., Gallaher, D., Weaver, R., and Van Geet, O. (2012). Green data center cooling: Achieving 90economization and unique indirect evaporative cooling. In *Green Technologies Conference, 2012 IEEE*, pages 1 –6.

[Xu and Liu, 2012] Xu, D. and Liu, X. (2012). Geographic trough filling for internet datacenters. In *INFOCOM*, pages 2881–2885.

[Xu et al., 2013] Xu, H., Feng, C., and Li, B. (2013). Temperature aware workload management in geo-distributed datacenters. In *Proceedings of the ACM SIGMETRICS/International conference on Measurement and modeling of computer systems*, pages 373–374.

[Yao et al., 2012] Yao, Y., Huang, L., Sharma, A., Golubchik, L., and Neely, M. (2012). Data centers power reduction: A two time scale approach for delay tolerant workloads. In *the 31st Annual IEEE International Conference on Computer Communications*, pages 1431–1439.

[Yigitbasi et al., 2011] Yigitbasi, N., Datta, K., Jain, N., and Willke, T. (2011). Energy efficient scheduling of mapreduce workloads on heterogeneous clusters. In *Green Computing Middleware on Proceedings of the 2nd International Workshop*, pages 1:1–1:6.

[Young et al., 2013] Young, B., Apodaca, J., Briceo, L., Smith, J., Pasricha, S., Maciejewski, A., Siegel, H., Khemka, B., Bahirat, S., Ramirez, A., and Zou, Y. (2013). Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment. *The Journal of Supercomputing*, 63(2):326–347.

[Zhang et al., 2013] Zhang, Q., Zhani, M. F., Raouf, B., and Hellerstein, J. L. (2013). Harmony: Dynamic heterogeneityaware resource provisioning in the cloud. In *The 33rd IEEE International Conference on Distributed Computing Systems*, pages 510–519.

[Zhang et al., 2012] Zhang, Y., Wang, Y., and Wang, X. (2012). Electricity bill capping for cloud-scale data centers that impact the power markets. In *Proceedings of the 41st International Conference on Parallel Processing*, pages 440–449.

[Zhou et al., 2008] Zhou, Y., Chakrabarty, D., and Lukose, R. (2008). Budget constrained bidding in keyword auctions and online knapsack problems. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1243–1244.

[Zhou and Naroditskiy, 2008] Zhou, Y. and Naroditskiy, V. (2008). Algorithm for stochastic multiple-choice knapsack problem and application to keywords bidding. In *Proceedings of the 17th International Conference on World Wide Web*, pages 1175–1176.

VITA

Hüseyin Güler was born in Istanbul, Turkey on March 30, 1988. He received his BSc. degree in Computer Science and Engineering from Bilkent University in 2011. In September 2011, he joined MSc. Program in Computer Science and Engineering at Koç University as a research and teaching assistant. He has been working on energy efficiency in large scale distributed systems and distributed interactive applications as part of the Networked and Distributed Systems Laboratory. He has co-authored three conference papers in EE-LSDS'13, W-PIN+NetEcon'13 (ACM SIGMETRICS) and IEEE/ACM DS-RT'13, and two journal papers (one under review, one ready for submission).