UNIVERSIDAD CARLOS III DE MADRID

TESIS DOCTORAL

STRUCTURAL ISSUES AND ENERGY EFFICIENCY IN DATA CENTERS

Autor: Jordi Arjona Aroca
Director: Antonio Fernández Anta, IMDEA Networks Institute

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

Leganés (Madrid), 13 de Febrero de 2015

UNIVERSIDAD CARLOS III DE MADRID

PH.D THESIS

STRUCTURAL ISSUES AND ENERGY EFFICIENCY IN DATA CENTERS

Author:	Jordi Arjona Aroca
Director:	Antonio Fernández Anta, IMDEA Networks Institute

DEPARTMENT OF TELEMATIC ENGINEERING

Leganés (Madrid), February 13th, 2015

Structural issues and energy efficiency in data centers

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Prepared by
Jordi Arjona Aroca, Universidad Carlos III de Madrid

Under the advice of
Antonio Fernández Anta, IMDEA Networks Institute

Departamento de Ingeniería Telemática, Universidad Carlos III de Madrid

---

Date: February 13th, 2015

Contact: jorge.arjona@imdea.org

This work has been supported by IMDEA Networks Institute.

TESIS DOCTORAL

STRUCTURAL ISSUES AND ENERGY EFFICIENCY IN DATA CENTERS

Autor:	Jordi Arjona Aroca
Director:	Antonio Fernández Anta, IMDEA Networks Institute

Firma del tribunal calificador:

Presidente:

Vocal:

Secretario:

Calificación:

Leganés, de de

Alea iacta est

JULIUS CAESAR,
before crossing the Rubicon

Acknowledgements

I had always thought that pursuing a Ph.D. was a matter of faith. Faith in yourself, in your work and faith in your capabilities. Nothing further from reality. Pursuing a Ph.D. has a lot of those things, but the most important thing is to have people behind you, people who support you and who help you to keep pushing.

There is a lot of people without whom this thesis would have been simply impossible. I want to start by thanking the IMDEA Networks Institute. To me it has been like an oasis in the desert, the way of doing a Ph.D. in the way a Ph.D. is done abroad without leaving Spain. It seems to me that doing it in any other place would have been definitely worse and much more difficult. Of course, IMDEA Networks is not just an institution, it is also people, and not just workmates but friends. These years would have not been the same without those soccer games, the spam, the cakes in the kitchen or the multilingual happy birthdays. Thanks Juan Camilo (I know you prefer just Camilo, but guess who is writing :)), great flatmate and better person; Ignacio and all his loud voice cursing, Héctor feeding us with nuts; Elli and Evgenia (we love you a bit, only a bit) and our fights for Antonio's time; Christian, Angelos, Vincenzo, Arash and Allyson, Thomas, Agustín, Luisfo, Philippe, Roderick, . . . I am sure I am forgetting someone, but you know you are important to me. Regarding supervisors and other members of IMDEA, I cannot forget Vincenzo Mancuso and all the times I have knocked on his door "d'ya have a sec??" even when he knew it was not going to be a sec. Thanks also to Joel and José Félix for your patience when we were just inventing how to do things. Thanks also to Joerg, Rade, Pierre, Brian, Flora. . . everyone.

I want to specially thank Lin Wang for being my bodyguard during my six months in Beijing, he did not have to fight for me but he was always there. Moreover, he proved himself as a great companion both at work and at leisure. Thanks Lin, you have a big share of merit in this thesis.

Although they will say they have not done anything I want to say in loud voice how proud I am of my parents, Antonio and Mari Rosa, who always supported me and my sister, gave us education, thanks to which I am writing these lines today, and backed and encourage us when we had to go far from them or abroad, because they knew it was the best for us. Thanks Dad and Mum. And also thanks to Silvia, my sister, because she has always been there, although I did not visit her in Germany and she came to China and India.

Talking about faith again, I have to talk about my supervisor, auxiliar father, friend, . . . He always had faith in me (or so he said :)). Thanks for pushing, for being a lighthouse in the dark.

I have already told you but I probably would have never chosen you if it had been my call when I joined IMDEA. It proves how stupid I was and shows that I have learnt something, if I were to choose now I would have absolutely no doubt. Thanks Antonio, for everything and for what it is to come.

Finally, I want to say thanks to my best friend, my greatest support, my everything, my wife. Thanks for your love and patience. For listening to me when talking about weird stuff. Thanks for being brave and encourage me to do what was best for me independently on how hard it was for you: going to Madrid to pursue my Ph.D., 9 months to the U.S., 6 months to China, 3 months to India, Ireland now and who knows what else. It would not have been possible without you supporting me. Thanks for completing me.

Abstract

With the rise of cloud computing, data centers have been called to play a main role in the Internet scenario nowadays. Despite this relevance, they are probably far from their zenith yet due to the ever increasing demand of contents to be stored in and distributed by the cloud, the need of computing power or the larger and larger amounts of data being analyzed by top companies such as Google, Microsoft or Amazon.

However, everything is not always a bed of roses. Having a data center entails two major issues: they are terribly expensive to build, and they consume huge amounts of power being, therefore, terribly expensive to maintain. For this reason, cutting down the cost of building and increasing the energy efficiency (and hence reducing the carbon footprint) of data centers has been one of the hottest research topics during the last years. In this thesis we propose different techniques that can have an impact in both the building and the maintenance costs of data centers of any size, from small scale to large flagship data centers.

The first part of the thesis is devoted to structural issues. We start by analyzing the bisection (band)width of a topology, of product graphs in particular, a useful parameter to compare and choose among different data center topologies. In that same part we describe the problem of deploying the servers in a data center as a Multidimensional Arrangement Problem (MAP) and propose a heuristic to reduce the deployment and wiring costs.

We target energy efficiency in data centers in the second part of the thesis. We first propose a method to reduce the energy consumption in the data center network: rate adaptation. Rate adaptation is based on the idea of energy proportionality and aims to consume power on network devices proportionally to the load on their links. Our analysis proves that just using rate adaptation we may achieve average energy savings in the order of a 30-40% and up to a 60% depending on the network topology.

We continue by characterizing the power requirements of a data center server given that, in order to properly increase the energy efficiency of a data center, we first need to understand how energy is being consumed. We present an exhaustive empirical characterization of the power requirements of multiple components of data center servers, namely, the CPU, the disks, and the network card. To do so, we devise different experiments to stress these components, taking into account the multiple available frequencies as well as the fact that we are working with multicore servers. In these experiments, we measure their energy consumption and identify their optimal operational points. Our study proves that the curve that defines the minimal power consumption of the CPU, as a function of the load in Active Cycles Per Second (ACPS), is neither concave nor purely convex. Moreover, it definitively has a superlinear dependence on the load. We also validate the accuracy of the model derived from our characterization by running different Hadoop applications in diverse scenarios obtaining an error below 4.1% on average.

The last topic we study is the Virtual Machine Assignment problem (VMA), i.e., optimizing how virtual machines (VMs) are assigned to physical machines (PMs) in data centers. Our optimization target is to minimize the power consumed by all the PMs when considering that power consumption depends superlinearly on the load. We study four different VMA problems, depending on whether the number of PMs and their capacity are bounded or not. We study their complexity and perform an offline and online analysis of these problems. The online analysis is complemented with simulations that show that the online algorithms we propose consume substantially less power than other state of the art assignment algorithms.

Table of Contents

List of Tables

List of Figures

Part I

Background

Chapter 1

Introduction

Internet has revolutionized our world. In 15 years it has passed from being hardly found in any home to be hardly not found in any pocket. We have become Internet-addicts and got used to continuously check our emails, videos, photos,... Through Google, Facebook, Twitter, we learn about what is happening with our friends or in any remote corner of the world, we are able to contrast news, we have access to any kind of information we are curious about. At the same time, Internet has also overturned the business world, simplifying and reducing the costs of sharing information in and between companies, and allowing any company, no matter how small it is, to have customers all over the world.

This has become real thanks to new concepts like the Internet of things, social networks, or cloud computing. However, at the end of the day, what Internet has done is putting huge amounts of data available to everyone. One of the keys of this availability of data has been the proliferation of data centers. Although some data centers are not necessarily large, like the ones usually deployed at many universities, companies or government institutions, large companies such as Google, Facebook, Amazon or Microsoft, among others, are building large scale data centers all over the world.

A large scale data center can be defined, in a nutshell, as an integrated facility housing a large amount of high end servers, up to the order of tens of thousands, interconnected by a dense network, hosting petabytes of data and consuming up to various tens of mega Watts. According to Belady [31], the building costs of a data center is between \$8M-\$30M/MW, being the average around \$20M/MW, depending on the kind of facility. These numbers lead to costs of $\$100-150$M for small/mid size facilities, while huge data centers, like Facebook's or Google's, can be in the order of \$600M.

However, although building a data center is expensive, they can be even more expensive to maintain. As we noted above, their average power consumption can be around 20MW per year, which unveils a second problem, their energy consumption. In a recent study, Van Heddeghem *et al.* [86] estimate that, between 2005 and 2012, worldwide aggregated data center energy consumption increased almost a 50%, reaching 270TWh from the previous 200TWh. This is roughly a

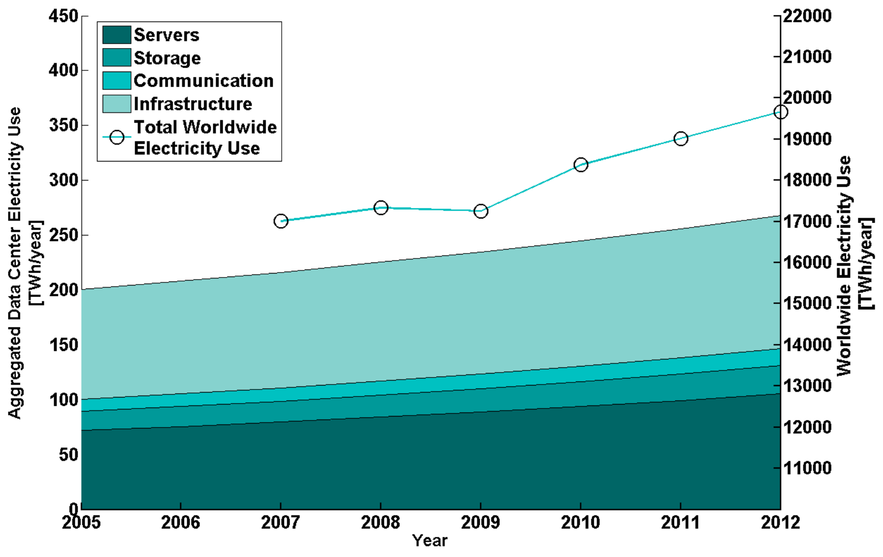


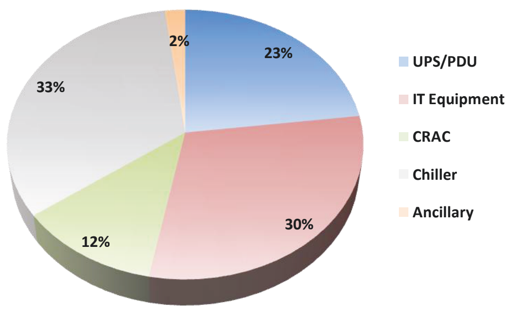
Figure 1.1: Worldwide use phase electricity consumption of data centers from 2005 to 2012 and the total worldwide electricity use from 2008 to 2012. Data center consumption is shown as the aggregation of its main consumers, servers, storage, communication and infrastructure.

1.5% of the total worldwide electricity consumption, and with a compound annual growth rate of a 4.4%.

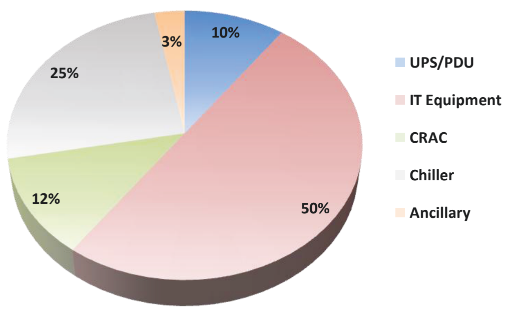
Therefore, it is easy to see why *greening* data centers has emerged as one of the main targets of the research community during the last years. *Greening* data centers has, in fact, a two-fold objective, reducing operation costs, i.e., saving money; and improving data center sustainability, i.e., reducing the amount of energy consumption attributed to data centers. In the same way, efficiency of several data center components, for instance cooling systems, may lead to a reduction on the building costs.

Data center research has become a broad field of research. Even when we restrain ourselves to reducing building cost or increasing energy efficiency, the complexity and variety of subsystems that can be found in a data center result in a huge amount of particular problems. Even if we restrain ourselves to two topics like optimizing the structural costs and increasing the energy efficiency of the different data center subsystems, the body of related work is overwhelming. In fact, if we look at some data provided by Barroso *et al.* [27, 28] we can see how the research in the field has contributed to change the energy consumption breakdown of data centers. In Figure 1.2 we can find the energy consumption breakdowns of a legacy data center with a PUE[1] value of around 2.0 in 2009 and 2013, in subfigures 1.2(a) and 1.2(b) respectively. Similarly, in subfigures 1.2(c) and 1.2(d), we can also see the evolution on the distribution of peak power usage

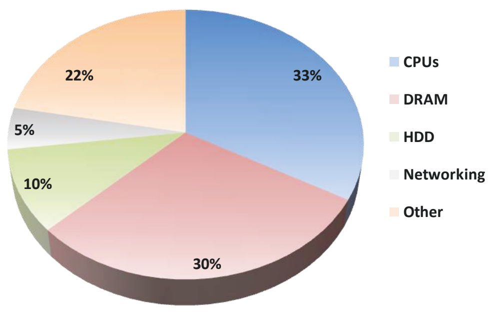
[1]PUE responds to *Power Usage Efficiency* and it is one of the most commons and broadly accepted efficiency metrics. It measures the amount of cooling power needed versus the amount of electricity to run the IT infrastructure. An ideal ratio is 1.0.



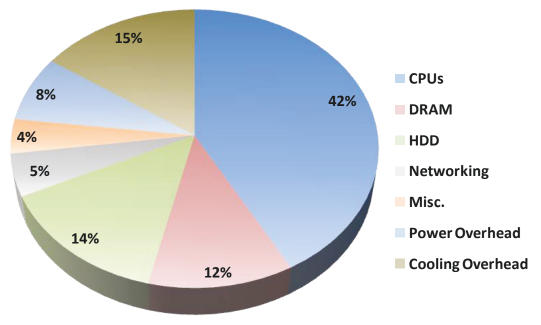
(a) Typical distribution of energy usage in a conventional datacenter with a PUE of 2.0 - 2009



(b) Typical distribution of energy usage in a conventional datacenter with a PUE of 2.0 - 2013



(c) Approximate distribution of peak power usage in a hardware subsystem in a Google's data center in 2007.



(d) Approximate distribution of peak power usage in a hardware subsystem in a Google's data center in 2012.

Figure 1.2: Evolution of the breakdown of a data center energy consumption and a hardware system between 2007-2013.

in a hardware subsystem in a Google's data center between 2007 and 2012. This evolution is the result of intense research in multiple fields concerning each one of the pieces of hardware, usage policies or interaction between them.

However, although this evolution on the power requirements can be extended to many data centers, as for instance the (each time more) energy proportional servers, they can not be applied to all of them. One of the variables that conditions the application of these latest techniques is, for instance, the size of the data center. Big companies proudly exhibit the very low PUEs of their flagship data centers, like Facebook's PrineVille and Luleå, with $1.06 - 1.08$ and 1.07 PUE, or Google's Hamina, with 1.14 PUE. However, the resources which can be devoted to the design and construction of these data centers are not the same devoted to smaller ones or by smaller companies. Similarly, these low PUEs are usually achieved because of some particular aspects of the location of the data center, like the use of Finland's gulf water in Hamina. According to the Uptime Institute [148], the average PUE is around $1.8 - 1.89$, which gives a better idea of how much energy efficiency can still be improved.

Table 1.1: Classification of Data Center types according to their size.

	Server Closet	Server Room	Localized Data Center	Mid-Tier Data Center	Enterprise Class Data Center
Size [sq ft]	< 200	< 500	< 1.000	< 5.000	> 5.000
# Servers (2005)	1.657.947	1.942.214	1.674.648	1.511.999	3.074.424
Estimated Energy consumption	11%	24%	21%	19%	24%
# Servers (est. 2009)	2.135.538	3.057.834	2.107.592	1.869.595	3.604.678

Moreover, although a bit outdated, Bayley *et al.* [22], in 2007, presented a study quantifying the amount of servers in different facilities according to their size. Some of the results of that study are presented in Table 1.1, showing the different categories, the estimated number of servers per category in 2005 as well as the distribution of energy consumption among them, and a prediction of the evolution of these numbers for 2009. These numbers show how most of the energy consumed is not necessarily in large enterprise data centers, but in smaller environments in which, in most cases, the PUE does not match the ones achieved by Facebook's or Google's flagship data centers. Nevertheless, it is important to remark that industry has become aware of this problem and more and more solutions are being provided each day to companies that only need small sized data centers, like Modular or Containerized Data Centers [9, 160] or integrated box solutions like IBM's Integrated Server Room [91]. In addition to this, Containerized Data Centers could even help to reduce the building costs of large data centers avoiding over built capacity and helping with over time scalability [152].

This thesis is divided in two parts, apart from the Background itself, one for each of the main topics that comprehend the different problems we addressed, these parts are Structural Issues and Understanding and Reducing Energy Consumption in Data Centers. This study intends to help to a better understanding of how energy is consumed in data centers as well as providing solutions which can be applied to data centers in any size range, from server closets to large enterprise data centers. We know provide some insights about what will be covered in these parts.

1.1 Structural Issues

One structural problem of data centers is the need of hosting more and more servers. Some traditional interconnection networks, such as meshes, toruses or grids were not an option given their poor scalability. More structured topologies, like trees or tree-like topologies also showed drawbacks that made difficult to scale them up. For instance, the number of ports per switch required by tree topologies was continuously increasing and so their price. At the same time, given its low bisection bandwidth, the upper tiers were easily congested even when the traffic at lower layers was rather low. Proposing new and better topologies capable to solve these problems became necessary.

One solution was building new topologies with smaller and cheaper switches in a larger quantity [10]. Fat-tree and fat-tree-based topologies were a step forward as they allow the use of these switches, reducing substantially the deployment costs of the network, as well as having larger bisection bandwidth. The benefits of using fat-trees led to multiple proposals like VL2 [78] or [131].

However, fat-trees also have some limitations when scaling up, and might end up depending also on large switches. For these reasons, new topology proposals tried to overcome fat-tree by presenting several improvements over it. Two of the most sound proposals were Dcell [80] and BCube [79]. These topologies were able to provide a larger number of redundant paths, reducing the congestion in the network, and better fault tolerance with similar price and energy consumption.

This first stage of studies about data center topologies had several benefits. First, and most obvious, meant an improvement over the existing topologies, allowing better scalability at similar or lower price, more resilient and less congested networks, helping to gather a broad set of parameters and metrics to compare the goodness of the different topologies. . . Also, it indirectly paved the way for a second wave of works which tackled the problem of energy consumption in data centers, taking advantage of the flexibility and goodnesses of these works.

We will deal with two problems in the context of structural issues. The first one is related to the set of parameters used to compare different network topologies. Two of these parameters, bisection width and bisection bandwidth, are broadly used to measure reliability and bandwidth of networks. We study these parameters for topologies which are, or can be, obtained from as product networks from smaller factor graphs. The second one is related to how the topologies are to be deployed in data centers. All the topologies that we have mentioned in this section share being more complex than most of traditional topologies. With our work we intend reducing the total wiring to be deployed to interconnect the different devices in a data center as well as reducing the average lengths of wires, what leads to less power leakages due to the inherent wire resistance as well as reducing latency.

1.2 Understanding and Reducing Energy Consumption in Data Centers

As we have already mentioned, improving the energy efficiency of data centers has become an issue of capital importance both for economic and of environmental reasons. Due to this importance, the amount of techniques that have been proposed to help in this area is so huge that it is impossible to present them in just one document. Hence, we now only introduce some of the techniques which are relevant for this thesis.

1.2.1 Rate Adaptation for Future Data Centers

We have talked about a first stage of studies about data center topologies in Section 1.1 and how they paved the way for a second wave of topological studies. Elastic tree [87] and PCube [89] are two examples of works which, based on some of the aforementioned topologies, namely fat-tree and BCube, present solutions to reduce the energy consumption of data centers networks. In particular, they propose controllers that can dynamically adjust the network structure depending on the amount of traffic. The result is a network were the number of available paths and active links and switches depends directly on the present load. These network controllers claim to be capable to save up to a 50% of the network energy costs of the networks they are based in.

Approaches like Elastic Tree and PCube rely in power-down strategies, i.e., switching off routing (switching) devices that are not needed if the traffic routing is optimized and the number of active paths in the network is adapted to the current load. One different and alternative approach is based on rate adaptation.

Rate adaptation was proposed by Nordman and Christensen [133] in 2005. Rate adaptation aims to achieve energy proportionality in the network by adapting the transmission rate of the network devices to the load present in their links. By doing this, the power consumption in the devices should be proportionally reduced and, hence, large energy savings could be achieved when the load in the links is low. Data center networks are known to have a very low average load, being quite low at least 60% of the time and experiencing traffic bursts during less that a 5%. Therefore, the savings in data center networks could be considerable.

However, although this technique has been widely studied, only in the very last years we have started to see devices that provide these capacities, such as InfiniBand [4] or the Cray Yarc [145] switch, which were some of the first switches to allow multiple speeds on their links and hence to start adapting their consumption to the associated link load. Nevertheless, there are still several limitations and problems that are expected to be solved by future devices.

One of the problems that we will discuss in this thesis is how, with the right network devices, rate adaption could be introduced in networking and in data centers, reducing the energy costs of data center networks.

1.2.2 How Servers Use Power

Servers are like puzzles where each one of its pieces has its own share of power consumption. At the same time, the global power consumption of a each one of these pieces is not constant, it depends on the stress we introduce in each one of them. Depending on the amount of accesses we do to disk, to memory, on the amount of data we send to or receive from the network and on the amount of processing we do or the heat we generate, the power required by Hard Drives, Memory, Network, CPU or cooling units will vary. There are also more components, as we saw in Figure 1.2, but those are usually assumed to be the major contributors to the power consumption of data center servers.

However, servers are not power proportional [26], i.e., the total power consumed is not proportional to the amount of load being processed, and usually, the main contributor to power consumption is the fact of having the machine switched on. In the recent past, the amount of power consumed by an idle machine compared to the power consumed when it worked at full speed could easily add up to a 70% of its total power consumption. If we focus in the last 7 years, we can consult the available public data from the SPEC power benchmark web [56]. Comparing results from the last quarter of 2007 against the last available ones (second quarter of 2014), we can see a reduction on the power consumption of servers when idle compared to its peak consumption from barely a 60% in most servers to roughly a $20 - 25\%$ of power consumption.

If we consider only the active range, i.e., the power variation between the idle state and the peak consumption, it has been traditionally assumed that most of the power is consumed by the CPU. Similarly to what happens with servers, processors do not consume power linearly, in proportion to the load. Although processor power consumption has usually been modeled in a linear fashion, everything changed with the arrival of multicore processors able to work at multiple frequencies. Multicore processors introduced multiple changes. First, cpus in the same processor are able to share on-chip and on-die resources, increasing,hence, the synergies and reducing power requirements [30]. Also, there are new parameters to be considered as variable voltages and frequencies that determine CPU speed and, therefore, power requirements. Due to this new complexity, being able to understand how servers consume power has become a must if we want to devise any technique or strategy that intends to reduce power or energy consumption.

In this thesis, we will present an empirical study were some of these aspects were analyzed and we were able to shed some light about the behavior of multicore and multifrequency machines based on real data. This knowledge can be applied in multiple techniques devoted to reduce the aggregated power and energy consumption of data centers. Not understanding the effect that placing a task in a server is going to have on its power consumption will result in non-optimal, in the best case, or in completely non-efficient, in the worst case, implementations of techniques such as speed scaling policies or virtualization strategies. We will now discuss about the latter two practices, speed scaling and virtualization, which are well known examples of techniques used to reduce the aggregated power and energy consumption in data centers.

1.2.3 Speed Scaling Based Techniques

Speed scaling is based on the ability of a processor to change its operating voltage and speed (frequency), and hence the speed and power consumption of the server. It is important to note that the values of voltage and frequency are not independent from one another. There is an intimate relation between them as, usually, the voltage conditions the range of available frequencies. In Table 1.2 we can find the different combinations of frequencies and voltages available for different processors.

One well known and extended implementation of speed scaling is *Dynamic Voltage and Frequency Scaling (DVFS)* which can be usually be found in the ample majority of servers which can be found nowadays in the market. DVFS can be configured with different governors or operating policies which will condition the way frequency adapts to the load in the system.

However, it is important to note that we can not just reduce the frequency as much as we want as it will affect the performance of the tasks being run in the machine. For this reason, usually, commercial implementations of DVFS have conservative policies whose main target is reduce operating frequency, and hence the consumption, of the machine when idle.

Table 1.2: Relation between frequency and voltage for different processors

Processors					
AMD Opteron 6276		*Intel Xeon W3530*		*Intel Xeon E5606*	
Voltages	Frequencies	Voltages	Frequencies	Voltages	Frequencies
0.9375V to 1.3125V	1.4 *GHz*, 1.6 *GHz*, 1.8 *GHz*, 2.1 *GHz*, 2.3 *GHz*, 2.3 *GHz*	0.750V to 1.350V	1.596 *GHz*, 1.729 *GHz*, 1.862 *GHz*, 1.995 *GHz*, 2.128 *GHz*, 2.261 *GHz*, 2.394 *GHz*, 2.527 *GHz*, 2.666 *GHz*, 2.793 *GHz*, 2.794 *GHz*	0.800V to 1.375V	1.2 *GHz*, 1.333 *GHz*, 1.467 *GHz*, 1.6 *GHz*, 1.733 *GHz*, 1.867 *GHz*, 2 *GHz*, 2.133 *GHz*

Most of the research in this field is devoted to find efficient policies which allow to reduce power consumption or energy consumption. It is important to remark the difference between both targets, let us give an easy example. Assume that we have a task that needs a time T to complete. If we reduce the operating frequency of the system, and hence the power consumption is reduced from C to C', it might happen that the task being run in our machine now needs a time T' to be run. If the total energy consumed $T \cdot C$ is larger than $T' \cdot C'$ we will have reduced the power consumption during a period of time, but spent more energy. This is neither good nor bad, both policies have their applications in different scenarios. However, we must remember that it is not trivial to optimize the power required and it is needed to carefully design the policies to be used.

1.2.4 Virtualization Based Techniques

We have already mentioned two important aspects of modern servers: that, with almost no exception, they are multicore and multifrequency servers, and that we pay a high cost in terms of power, just for having them idle, (i.e., powered on but not doing any task). However, think now, just for a second, that, years ago, running multiple tasks in a machine was through multithreading, i.e., running them in parallel with no isolation. When a task required some isolation, for security or other reasons, it had to be run alone in a server, what implies that the server resources not being used by that task were wasted. Keep in mind also that, in old servers the percentage of power used just for having them powered on was larger than nowadays. Additionally, there were some problematic situations with multithreading, as the existence of resource-greedy users or tasks. In order to tackle this situation we use *Virtualization*. Although virtualization was originally developed in the 1960s by IBM, it was forgotten and then recovered again in the 1990s. We can define virtualization as *"a technology that combines or divides computing resources to present one or many operating environments using methodologies like hardware and software partition-*

ing or aggregation, partial or complete machine simulation, emulation, time-sharing, and many others" [51]. We call each one of these operating environments a *Virtual Machine (VM)*. Hence, instead of running tasks in a per server basis or use multithreading sharing the resources pool, virtualization allows us to run tasks in a per VM basis, therefore running multiple tasks (with limited resources) independently in the same server.

Nevertheless, virtualization only opened the door for future improvements in how to reduce the power consumption of data centers from a server perspective. Two of these consequences were consolidation and virtual machine allocation techniques.

Consolidation is probably the most straightforward consequence of virtualization. Since we gained the ability of putting multiple virtual machines in one server it is logic trying to maximize the efficiency of servers. Consolidation aims to either maximize the aggregated number of tasks being run keeping a constant number of active servers, or minimize the number of servers needed to run a set of tasks. In both cases, the contribution of virtualization to increase the productivity and the energy efficiency of data centers is clear.

Similarly, and intimately related with consolidation, we have virtual machine allocation techniques. Given that the assignment of virtual machines to servers is an NP-hard problem (it can be easily reduced to problems such as bin packing or 3-partition for instance, as we will see in Chapter 7), multiple heuristics and algorithms have been devised to tackle the online version of the problem. Algorithms like First Fit, Packing, Most (Least) Loaded First . . . try to obtain the best assignment of tasks to servers according to a certain magnitude, like cost, energy consumption. . . although in general try to minimize the number of active servers.

However, most of these algorithms are based on linear models for the power consumption of cores. Based on the insights we got with the characterization of a data center server we will study the effect of assuming a non-linear power consumption model for data centers servers. Based on this model, we will perform a competitive analysis of different VM to physical machine power aware assignment strategies under different hypothesis, like the capacity or the number of available physical machines. We will propose our own strategies and compare their power and energy consumption with the ones of some of the aforementioned algorithms (first fit, packing. . .).

1.3 Overview and Summary of Contributions

In the previous section we have enumerated several problems and challenges related to data centers. However, although each one of them are broad research topics, they are only a small subset of the challenging problems that are to be solved in such an ecosystem. We know present the problems in which we worked and the contributions we made for each one of them.

Starting with data center topologies, we found that one of the most important parameters used to characterize and compare these newly proposed topologies was the bisection width and, intimately related to it, the bisection bandwidth. This relevance comes from the fact that bisection (band)width bounds the speed at which information can be moved from one side of a network to another. Similarly, we found that many topologies, old and new, could be seen as a product graph, i.e., as the result of combining smaller graphs with the Cartesian product operation. Inspired by the need of a solution for the bisection width of the multidimensional tori, which was posed by Leighton [105, Problem 1.281] and had remained as an open problem for more than 20 years, we devised a technique to lower and upper bound the bisection width of product graphs. This technique can be successfully applied to many classical interconnection networks obtained by the application of the Cartesian product of graphs to obtain an exact result for their bisection width and bisection bandwidth. Using this technique we obtained an exact result for both the bisection width and bandwidth of the multidimensional tori, products of extended complete binary trees [65] and rings, and products of mesh connected trees (a.k.a. products of complete binary trees), and paths. Moreover, we show that our technique can also be applied to newly devised and more complex topologies, such as BCube, for which we obtain bounded results for its bisection width and bisection bandwidth.

The second problem we tackled is related to the increasing complexity of most of the lately proposed data center topologies. This is partially due to the need of placing more and more nodes, partially because of the implementation of properties that improve different aspects of the network. Obviously, these networks need to be deployed and this deployment is not trivial. A suboptimal deployment leads to an excess in the associated costs and a worse performance. This excess in the costs is represented, for instance, by deploying extra wiring. Having extra wiring results in more costs in buying the wires, more consumption due to the losses associated to the extra length of the wires, augmented latency (this aspect might seem irrelevant, but there are environments where nanoseconds matter, like in high frequency trading [36, 52]), or simply by having a more complex wiring scheme which is more costly to maintain. Given that a data center network is basically a graph which consists of servers and interconnection devices, and that the set of racks in a data center can be seen as a 3-dimensional array, the problem of allocating the servers of a data center into its racks can be seen as an instance of the *Multidimensional Arrangement Problem (MAP)* [85], which is NP-hard. To solve this problem, we have devised JAM, a Tabu-based two-stage simulated annealing algorithm for the MAP. This algorithm is able to match or approximate most of the best and optimal solutions from a set of 81 benchmark instances taken from the minimum Linear Arrangement (minLA) and Quadratic Assignment Problem (QAP) literature. Adapting these instances from their original configuration to 1, 2 or 3 dimensions, we have also broadened the available instances for minLA and QAP, as well as created a set of benchmark instances in multiple dimensions for MAP.

Switching to the field of energy consumption in data center networks, we also studied how to reduce the energy consumed by interconnection devices. We present a solution based on rate adaptation for data center networks (although nowadays hardware is still limiting the potential gains that could be achieved by implementing rate adaptation). Assuming that the latter was feasible and that we have switches that have a set of possible rates at which they can operate, we show that savings of up to a 40% can be achieved by simply moderating the rates of switches.

Moreover, these savings can be increased by combining our technique with power-down policies, i.e, reducing the number of active network devices if possible.

Our next contribution is related to how energy is spent in physical machines. As we mentioned in Subsection 1.2.3, in order to device wise consolidation algorithms or speed scaling policies for our systems, the first step is to properly understand how power is consumed in our physical machines. With the irruption of multicore servers we find a certain inconsistency in the literature as it is usual to find works that assume linear models of consumption for servers, while some others assume models different from linear (such as superlinear models). To shed some light in this problem, we performed an empirical study with 3 different servers of different architectures (Intel Xeon and AMD Opteron). Our first contribution was showing that the metric used to express load matters, and, hence, using relative magnitudes, like load percentage, might lead to deceiving results. We introduced the Active Cycles Per Second (ACPS) metric, which is an absolute magnitude, related to the frequency of operation, and that denotes the amount of computer cycles used to process load by a server per second. Our study analyzed the contribution to the total power consumption of three different components of data center servers, namely, CPU, disks, and network, and their dependencies in certain parameters, like the frequency. The most important contribution of this work is showing that the curve that defines the minimal CPU power as a function of the load is neither linear nor purely convex as has been previously assumed. Similarly, we also study the effects of the operating frequency and other parameters in the power consumption of disks and network. We validate our model by means of computing the PageRank metric of a graph and a WordCount application in a Hadoop platform, first without network activity, next with bulky network activity, and finally with a two-server cluster. We find that the energy can be estimated with an error that is below 4.1% on average and never worse than a 10%.

One of the conclusions of the previous work is that there exists an optimal point of operation distinct from the 100%. This means, in simple words, that running a server at full load and full speed, is not always optimal. This also contradicts some statements that have been traditionally assumed to be true regarding how much load has to be processed by a server.

Based on the latter study and the conclusions which can be extracted from it, a question that immediately follows is how to assign virtual machines to physical machines in a power efficient way. Our last contribution consists of an analytical study of what we call the Virtual Machine Assignment (VMA) problem. We study this problem from different points of view, imposing, or not, restrictions on the number of physical machines or on their CPU capacity. Namely, we study 4 models, $(\cdot,\cdot)$-VMA (no restrictions in CPU capacity or number of servers), $(C,\cdot)$-VMA (CPU capacity is finite, number of servers is infinite), $(\cdot,m)$-VMA (CPU capacity is finite, number of servers is bounded) , and (C,m)-VMA (both CPU capacity and number of servers are finite). We show that the decision version of the (C,m)-VMA problem is strongly NP-complete. We show as well that the $(C,\cdot)$-VMA, $(\cdot,m)$-VMA and $(\cdot,\cdot)$-VMA problems are strongly NP-hard. Hence, there is no FPTAS for these optimization problems. We also show the existence of a PTAS that solves the $(\cdot,\cdot)$-VMA and $(\cdot,m)$-VMA offline problems. On the other hand, we prove lower bounds on the approximation ratio of the $(C,\cdot)$-VMA and (C,m)-VMA problems. With respect to the online version of these problems, we prove upper and lower bounds on the competitive ratio of the $(\cdot,\cdot)$-VMA, $(C,\cdot)$-VMA, $(\cdot,m)$-VMA, and (C,m)-VMA problems. Finally, we compare our algorithm to a modified version to other real approaches and show its advantages by simulation.

1.4 Roadmap

The rest of the thesis is structured as follows. We start with a Background part, including this Introduction Chapter and Chapter 2, which presents an overview of the state of the art for each one of the problems we have mentioned. The rest of the document in divided into two parts, Part II, Structural Issues, and Part III, Understanding and Reducing Energy Consumption in Data Centers. Part II starts with Chapter 3, a study about how to compute the bisection (band)width of different product networks. Also in this part, we find Chapter 4, which presents the design and application of a simulated annealing heuristic to reduce costs in data centers. Already in Part III, we will start, in Chapter 5, by studying how to reduce energy consumption in data center networks by implementing rate adaptation. Afterwards, we will be presenting a characterization of how energy is consumed in a physical machine by 3 of its main components, namely, CPU, disk and network. We will conclude Part III with an analytic study about how virtual machines should be assigned to physical machines in order to reduce the energy consumption in data centers and propose different algorithms tackling this problem. Finally, Chapter 8 concludes this thesis with a summary of our main results, a discussion on the implications of these results and future research directions.

This thesis covers contributions from the following literature:

- **Jordi Arjona Aroca** and Antonio Fernández Anta. “Bisection (Band)Width of Product Networks with Application to Data Centers”. *IEEE Transactions on Parallel and Distributed Systems* in vol. 25, issue 3, March 2014.

- **Jordi Arjona Aroca**, Antonio Fernández Anta. “JAM, A Tabu-based Two-Stage Simulated Annealing for the Multidimensional Arrangement Problem”. *Hybrid Metaheuristics 2014*, 155-168.

- Lin Wang, Fa Zhang, Chenying Hou, **Jordi Arjona Aroca**, Zhiyong Liu. “Incorporating Rate Adaptation into Green Networking for Future Data Centers”. *2013 IEEE 12th International Symposium on Network Computing and Applications, NCA’13*, Cambridge, MA, USA, August 22-24, 2013. IEEE 2013.

- **Jordi Arjona Aroca**, Angelos Chatzipapas, Antonio Fernández Anta and Vincenzo Mancuso. “A Measurement-based Analysis of the Energy Consumption of Data Center

Servers". *International Conference on Future Energy Systems (ACM e-Energy) 2014*, 63-74. June 2014.

- **Jordi Arjona Aroca**, Antonio Fernández Anta, Miguel A. Mosteiro and Christopher Thraves. "Power-efficient Assignment of Virtual Machines to Physical Machines". Workshop on Adaptive Resource Management and Scheduling on Cloud Computing (ARMS-CC).

Additionally to the above, the following papers with related content have been published during the development of this thesis:

- **Jordi Arjona Aroca**, Antonio Fernández Anta, Miguel A. Mosteiro and Christopher Thraves. "Power-efficient Assignment of Virtual Machines to Physical Machines". *Accepted for Publication in Future Generation Computer Systems Journal (FGCS), Special issue on "Advanced Topics in Resource Management for Ubiquitous Cloud Computing: an Adaptive Approach".*

- Lin Wang, Fa Zhang, **Jordi Arjona Aroca**, Athanasios V. Vasilakos, Kai Zheng, Chenying Hou, Dan Li, Zhiyong Liu. "GreenDCN: a General Framework for Achieving Network Energy Efficiency in Data Centers". *IEEE Journal on Selected Areas in Communications* in vol. 32, no. 1, January 2014.

- **Jordi Arjona Aroca**, Antonio Fernández Anta, Miguel A. Mosteiro and Christopher Thraves. "Power-efficient Assignment of Virtual Machines to Physical Machines". *XXI Jornadas de Concurrencia y Sistemas Distribuidos (JCSD 2013)*, 19-21 June 2013, San SebastiÃ¡n, Spain.

- **Jordi Arjona Aroca** and Antonio Fernández Anta. "Bisection (Band)Width of Product Networks with Application to Data Centers". *9th Annual Conference on Theory and Applications of Models of Computation, TAMC 2012*, May 16-21, 2012, Beijing, China.

- **Jordi Arjona Aroca** and Antonio Fernández Anta. "Bisection Width of Multidimensional Product Graphs". *Young Researchers Forum (YRF 2011)*, in conjunction with the 36th International Symposium on Mathematical Foundations of Computer Science (MFCS 2011), 22 -26 August 2011, Warsaw, Poland.

Chapter 2

Related Work

2.1 Bisection Width and Bisection Bandwidth of Product Networks

2.1.1 Background

The bisection width and the bisection bandwidth of interconnection networks have always been two important parameters of a network. The first one reflects the smallest number of links which have to be removed to split the network into two equal parts, while the second one bounds the amount of data that can be moved between these parts. In general, both values are derivable one from the other, which is the reason why most previous work has been devoted to only one of then (in particular, the bisection width).

The bisection width has been a typical goodness parameter to evaluate and compare interconnection networks for parallel architectures [57, 61, 105]. This interest has been transferred to the Network-On-Chip topologies, as the natural successors of the parallel architectures of the 90's [97, 118, 144, 172]. The bisection (band)width is also nowadays being used as a reference parameter on the analysis of the topologies that are being deployed in data centers. The bisection bandwidth can be used to compare the potential throughput between any two halves of the network in different topologies. Similarly, the bisection width also gives some insights on their fault tolerance, showing the maximum number of critical link errors a network can suffer before being split into two halves. This can be seen in recent papers which propose new topologies, like BCube [79] or DCell [80]. The bisection (band)width is one of the parameters used to compare these new topologies with classical topologies, like grids, tori, and hypercubes, or with other datacenter topologies, like trees and fat trees.

Finding the exact value of the bisection width is hard in general. Computing it has proven to be challenging even for very simple families of graphs. For instance, the problem of finding the exact bisection width of the multidimensional torus was posed by Leighton [105, Problem 1.281] and has remained open for 20 years. One general family of interconnection networks, of which the torus is a subfamily, is the family of product networks. The topology of these networks is obtained by combining factor graphs with the Cartesian product operator. This technique allows to build large networks from the smaller factor networks. Many popular interconnection networks are instances of product networks, like the grid and the hypercube. This, however, is not only a characteristic of classical interconnection networks. Some of the recently proposed data center topologies also share the property of being constructed by combining basic instances, as the already mentioned Bcube [79] and Dcell [80] or HCN and BCN [81].

Summarizing, in the context of data centers, studying the bisection width and bandwidth of product networks will help us to compare some of the most relevant drawbacks and goodnesses of newly proposed and old topologies, such as their reliability or bandwidth.

2.1.2 Related Work

To our knowledge, Youssef [169, 170] was among the first to explore the properties of product networks as a family. He presented the idea of working with product networks as a divide-and-conquer problem, obtaining important properties of a product network in terms of the properties of its factor graphs.

The bisection width of arrays and tori was explored by Dally [58] and Leighton [105] in the early 90s, presenting exact results for these networks when the number of nodes per dimension was even. The case when there are odd number of nodes per dimension was left open. Rolim *et al.* [141] gave the exact values for the bisection width of 2 and 3-dimensional grids and tori, but left open the question for higher number of dimensions.

For the special case in which all the factors are isomorphic, Efe and Fernández [64] provided a lower bound on the bisection width of a product graph as a function of a new parameter of a factor network they defined, the maximal congestion. Nakano [127] presented the exact value of the bisection width for the Cartesian product of isomorphic paths and cliques (i.e., square grids and Hamming graphs). If the factor graphs have k nodes, he proved that the d-dimensional square grid has bisection width k^{d-1} when k is even, and $\frac{(k^d-1)}{(k-1)}$ when k is odd. Similarly, the square Hamming graph has bisection width k^{d+1} when k is even, and $(k+1)\frac{(k^d-1)}{4}$ when k is odd. The exact bisection width of the d-dimensional square grid was found independently by Efe and Feng [63].

For Chapter 3, the work of Azizoglu and Egecioglu is very relevant. In [19] and [21] they studied the relationship between the isoperimetric number and the bisection width of different product networks. In the former paper, they find the exact value of the bisection width of the cylinders (products of paths and rings) with even number of nodes in its largest dimension. In the latter reference they found the exact bisection width of the d-dimensional grid $A^{(d)}_{k_1,k_2,\ldots,k_d}$, with k_i nodes along dimension i, and where $k_1 \geq k_2 \geq \ldots \geq k_d$. The value of this bisection width is $BW(A^{(d)}_{k_1,k_2,\ldots,k_d}) = \sum_{i=1}^{\alpha} C_i$, where α is the smallest index for which k_i is even ($\alpha = d$ if no index is even), and $C_i = \prod_{j=i+1}^{d} k_j$.

2.2 A Simulated Annealing Approach to the Multidimensional Arrangement Problem

2.2.1 Background

Assignment and arrangement problems have been extensively studied for decades. The most classical and wellknown application of these problems is the assignment of n facilities to m locations in order to minimize or maximize a certain magnitude, such as cost, flow, etc. In this section, we introduce one of these arrangement problems, the Multidimensional Arrangement Problem (MAP), which was firstly studied by Hansen [85]. MAP covers a great number of applications, such as graph drawing or job scheduling (in 1 dimension), the backboard wiring problem or the arrangement of electronic components in printed circuits (in 2 dimensions), and placing servers in the racks of a data center (in 3 dimensions).

MAP can be defined as follows. Given a graph $G = (V, E)$, a host D-dimensional array $H(V', E')$ such that $|V'| \geq |V|$, we can define the ***Multidimensional Arrangement Problem*** as the embedding of G into H, i.e., a mapping of the edges of G to paths in H, such that the aggregated length of the paths in H is minimized. As we will usually work with weighted graphs, the goal of MAP is minimizing the weighted sum of the path lengths. Formally, the cost of an embedding $\varphi : V \to V'$ is defined as

$$C(\varphi) = \sum_{(u,v)\in E} w_{uv} \cdot dist\left(\varphi(u), \varphi(v)\right), \tag{2.1}$$

where w_{uv} is the weight of edge (u, v) and $dist\left(\varphi(u), \varphi(v)\right)$ is the Manhattan distance (the path length) between the images of u and v in the host graph H.

The particular case of $D = 1$ is a well known problem, called the ***minimum linear arrangement (minLA)***. In this problem, the objective is to embed a graph onto a one dimensional array. As minLA is known to be NP-complete and MAP has minLA as a special case, it can be concluded that MAP is NP-hard.

On the other side, we have *Simulated annealing (SA)*, a local search based metaheuristic, introduced by Kirkpatrick *et al.* [99] in 1983. It was inspired in the metallurgical process of annealing, and used to solve combinatorial optimization problems. An SA algorithm is usually described by the following elements: initial solution, neighborhood function, cooling rate, number of iterations per temperature, and stop criteria or final temperature. In a nutshell, SA applied to MAP starts from an initial solution φ_0; and then, in each iteration, a candidate neighboring solution φ_l is chosen, based on a cost-based neighborhood function. Once φ_l is chosen it is compared against the current solution (φ^*) and, depending on whether $\delta = C(\varphi_l) - C(\varphi^*)$ is larger than 0 or not, φ_l is accepted as the new current solution φ^* or tested with an acceptance function. This function depends on the current temperature and is based on the Metropolis criterion [115], that will finally accept φ_l as the new φ^* or refuse it. If a new solution is chosen and it is better than the best-so-far solution φ_{best}, it becomes the new φ_{best}. After running a given number of iterations the system's temperature is cooled down. This process follows until a total number of iterations is run or a termination criteria is met.

Observe that the acceptance function allows the heuristic to admit solutions which are worse than the previous ones. This is generally known as climbing up and helps to avoid that heuristics are trapped in a local optimum. Although the mechanics of SA are not complicated, choosing the cooling rate, stop criteria, and neighborhood function is not trivial.

2.2.2 Related Work

The *Quadratic Assignment Problem,* which is a more general problem than MAP, is an NP-hard problem [143] which has been creating interest during more than 50 years [100]. The QAP objective function can be mathematically formulated as follows

$$\sum_{i=1}^{n}\sum_{j=1}^{n} f_{ij} \cdot dist(\pi(i)\pi(j)) + \sum_{i,\pi(i)} b(i,\pi(i)),$$

where f_{ij} is the flow between facilities i and j, $\pi(\cdot)$ is the location at which a facility has been assigned, $dist(x, y)$ denotes the distance between two locations x and y, and $b(i, x)$ is the initial allocation cost of facility i to location x. Many well-known problems, like the traveling salesman problem (TSP), minLA, and MAP, are special cases of QAP.

Some exact algorithms have been developed to solve the QAP problem. However, they are only capable to solve small instances due to the enormous computation capacity required. The largest instances solved optimally surpassed just recently the 100 locations frontier [70], but most of the latest works still work with instances of 30-40 locations [70] [134]. These algorithms typically use branch and bound, branch and cut, or dynamic programming.

Approximate methods have also been developed to tackle the QAP problem. We classify them in heuristics and metaheuristics. Starting with heuristics, most of the ones that have been developed can be grouped in constructive, enumeration, and improvement methods. We can find some examples of heuristics applied to the QAP problem in [72, 117, 132].

Despite of the richness in heuristics, metaheuristics have been attracting most of the attention lately. Most of the metaheuristics applied to the QAP problem can be included in one of the following families: genetic algorithms (GA) [60, 120], simulated annealing (SA) [42, 167], ant colony optimization (ACO) [149], tabu search (TS) [95, 121, 122, 150], breakout local search (BLS) [34], greedy randomized adaptive search procedures (GRASP) [107], variable neighborhood search (VNS) [171], or hybrid combinations of them [71, 151]. Given that QAP is more general than MAP, it is possible to adapt many of these techniques to obtain solutions also for MAP.

Simulated annealing was one of the first techniques applied to the QAP problem (c.f., Burkard *et al.* [42], Wilhelm *et al.* [167]). We now describe some of the main characteristics of some of the latest works using SA, alone or combined with other techniques. We start with the work of Wang [161], who proposed in 2007 a tabu-based simulated annealing algorithm. In that work, a pure SA algorithm was compared to a tabu-search SA, trying different tabu list sizes and also trying different guided restart and reannealing strategies, enhancing the ability to escape from local optima. In 2012, Wang [162] presented a new work based also on simulated annealing, but trying different guided restart strategies. In both works a local-search-based neighborhood function was used jointly with a geometrical cooling rate schedule (like Kirkpatrick *et al.* [99]), reheating the algorithm when a restart takes place. In 2012, Jingwei *et al.* [98] presented a new hybrid algorithm combining ant colonies and simulating annealing. Here, simulated annealing was used to select the best ants in each iteration, while the cooling schedule was also geometrical. In 2003, Misevičius [119] presented a very detailed work comparing multiple previously proposed cooling schedules. With this, he proposed an SA heuristic using a normal-local-search-based neighborhood function, an inhomogeneus annealing cooling schedule without equilibrium tests, like the one proposed by Connolly *et al.* in [55], and modified reannealing so the cooling schedule oscillates depending on the behavior of the annealing. This heuristic was completed by a post optimization stage based on Taillard's robust tabu search. This heuristic was even able to improve one of the QAPLIB [41] instances.

Finally, Tello *et al.* [139], in 2008, presented a 2-stage simulated annealing algorithm for the minLA problem. This work was able to improve multiple results from the typical set of minLA benchmarks compiled by Petit [137]. Its main contribution is to design a 2-stage SA algorithm, where the first stage obtains an initial approximation through a frontal increase minimization algorithm, and the second stage is devoted to improve this initial solution. They consider a modified median-based neighborhood function in which the typical 2-exchange strategy is conditioned by the nodes connected to a candidate-to-be-moved node. They also consider different ways of establishing the initial temperature, based on [154], and a different cooling schedule [7]. We will detail these aspects when describing our algorithm in Chapter 4, as we adopted and adapted some of their ideas for our MAP heuristic.

2.3 Rate Adaptation and Green Data Center Networking

2.3.1 Background

As we showed in Figure 1.2 in Chapter 1, the contribution of IT equipment to the total power requirements of data centers has increased during the last years. Although most of this power is consumed by servers, another relevant contributor to these expenses is the network equipment. Moreover, given the improvements in the proportionality of servers, the power consumed by networks is becoming more relevant and should not be ignored. Otherwise, the power consumed by networks will become a first order operating expenditure in future data centers. Just as an illustrative example of the magnitudes we are referring to, it has been reported in [6] that the total power consumed by network elements in data centers in 2006 in the U.S. was about 3 billion kWh and it is still increasing rapidly. Therefore, making the network energy proportional can provide considerable energy savings and tremendous economic benefit.

Traditional data center networks are mostly structured as a 2N tree topology [2]. In a 2N tree, the effective bisection bandwidth can be easily cut down by a small number of failures. Alternatives such as FatTree [10, 131], VL2 [78], BCube [79] and DCell [80] provide much richer connectivity and can handle failures more gracefully. However, the static provisioning irrespective to real workloads forces most of the designs into a dilemma: high connectivity comes with high power consumption, as currently used network elements are unlikely energy proportional. It has been verified in [114] that 60% of the time, the average traffic stays quite small, and the portion of time in which traffic peaks is less than 5%. As a result, the power consumption of the network should be proportional to the workload to achieve energy conservation.

The topic of greening the data center network has been widely explored. Most of the work can be categorized into two groups in general. The most straightforward way is designing energy-efficient topologies which can provide similar connectivity while using less network devices [8, 90]. However, the potential of this approach is limited since we have to guarantee sufficient bandwidth for traffic bursts. Another option consists in reducing the amount of active devices in current networks. This is generally accomplished by consolidating traffic flows and turning off unnecessary devices (e.g. [87, 111, 146, 165]). The key observation behind this line is the connectivity redundancy and the traffic load variation in current data center networks. However, when we switch off devices, the network topology will be changed. Since this topology transformation cannot be completed in a short time, the network may suffer from oscillation. Consequently, maintaining the quality of service in the network will become tricky.

We propose to incorporate rate adaptation into data center networks to achieve energy conservation. To the best of our knowledge, this approach has not been deeply explored before. Rate adaptation was proposed by Nordman and Christensen [133] and has been widely studied. The main idea of rate adaptation is to approach energy proportionality by varying the link rate adaptively to meet its carried load. Since being proposed, rate adaptation starts to be supported by some production devices, such as InfiniBand [4]. As shown in Table 2.1, an InfiniBand link is a serial link with single or multiple lanes, each of which can operate at one of five data rates. With a lower data rate, the power consumption is accordingly smaller. Also the ADSL2+ standard [77] and the Cray YARC [145] switch allow links to be configured to specific speeds. Although there are still some limitations in applying rate adaptation directly, we believe that future network devices will provide a wide set of operating rates in order to keep up with the green computing trend. In this sense, this work also reveals the potential of saving energy by using rate adaptation in future data center networks.

We also emphasize the opinion that a good energy saving solution comprises not only single-device energy proportionality, but also network-wide optimization. The basic idea is to globally optimize the scheduling and routing of flows and dynamically adjust the rates of network devices according to the loads. Based on rate adaptation, we aim at exploring efficient routing algo-

Table 2.1: Effective theoretical data rate of InfiniBand

Rate	SDR	DDR	QDR	FDR	EDR
1X	2Gbit/s	4Gbit/s	8Gbit/s	13.64Gbit/s	25Gbit/s
4X	8Gbit/s	16Gbit/s	32Gbit/s	54.4Gbit/s	100Gbit/s
12X	24Gbit/s	48Gbit/s	96Gbit/s	163.64Gbit/s	300Gbit/s

rithms for improving the energy efficiency in data center networks. Specifically, in Chapter 5, we model this rate-adaptive energy-efficient routing problem and provide a constant approximation algorithm to solve it. This is the most significant difference compared with the most relevant work [157] which provides an energy-efficient traffic engineering solution for general networks by utilizing rate adaptation heuristically.

2.3.2 Related Work

We can, basically, group the related work around two main topics, green networks and energy-efficient data center networks.

Significant amount of work has been done on energy-efficient and green networking. Gupta *et al.* [83] proposed the idea of reducing the overall energy for Internet, and then studied an energy saving problem for LAN (Local Area Network) [84], suggesting to utilize the multiple speed states of Ethernet interfaces. ALR (Adaptive Link Rate) was proposed by Nordman and Christensen [133] for achieving energy proportionality on Ethernet network links. Combining sleeping and rate adaptation, Nedevschi *et al.* [129] studied how to reduce the network energy consumption during network idle or low-load period. In order to achieve network-global energy efficiency in Internet, Andrews *et al.* [13] then devised an effective approximation algorithm for a centralized network-wide routing model, assuming speed scaling capabilities for all network elements. They also studied a similar problem by powering down network devices [16]. In a subsequent work, Andrews *et al.* [15] studied a flow scheduling problem for saving energy in an adversary network environment, while maintaining the stability in the meantime. Cianfrani *et al.* [53] proposed energy-aware OSPF routing solutions by integrating the energy-saving strategy into IP routing protocols. To achieve scalability, Vasic *et al.* [157] come up with a heuristic energy-aware traffic engineering solution where multiple operational rates of network devices are exploited. Later, they proposed REsPoNse [156], where energy-critical paths are used to handle the optimality-scalability trade-off in the pursuit of power conservation in networks.

On the other hand, prior work on energy-efficient data center networks can be classified into two categories. The first one tries to find some optimization methods for the currently used network topologies. One representative work in this category is the *ElasticTree* by Heller *et al.* in 2010 [87]. In this work, a network-wide power manager is proposed to dynamically adjust the set of active network devices to follow changing traffic in data centers. Shang *et al.* [146] considered routing with as few network devices as possible. They modeled this energy-aware routing problem and then gave heuristics to solve it. Mahadevan *et al.* [111] discussed how to reduce the network operational power in large scale systems and data center networks. Recently Wang *et al.* [165] proposed CARPO, a correlation-aware power optimization algorithm that considers the correlations between different flows.

The second category tries to design better network topologies to reduce the power cost of the whole network. Abts *et al.* [8] proposed a new kind topology called flatted butterfly which can reduce the number of switches, bringing a big amount of energy savings, while guaranteeing a similar connectivity. They also discussed how to provide dynamic link rates to meet the real link loads. Huang *et al.* [90] provided another kind of server-centric network topology which can vary bandwidth availability based on traffic demands.

Differing from others, we will consider a new rate-adaptive energy-saving model in data center networks. With the underlying trend of rate adaptive network devices in mind, we discuss in Chapter 5 how to provide green networking for future data centers.

2.4 Characterizing the Energy Consumption of Data Center Servers

2.4.1 Background

While in Section 2.3.2 we referred to the relevance of network elements regarding energy consumption in data centers, we now concentrate our attention on the characterization of data center servers and the energy they consume.

Indeed, although many energy saving techniques have been proposed during the recent years, such as virtualization plus consolidation or scheduling optimization [102, 126], in order to obtain full benefit of them it is crucial to have a good characterization of the servers in the data center, as a function of the utilization of the server's components. That is, it is necessary to know and understand the energy and power consumption of servers and how this changes under the different configurations. There is a large body of literature on characterizing servers' energy and power consumption. However, the existing literature does not jointly consider phenomena like the irruption of multicore servers and dynamic voltage and frequency scaling (DVFS) [166], which are key to achieve scalability and flexibility in the architecture of a server. With these new parameters, more variables come into play in a server configuration. Learning how to deal with these new parameters and how they interact with other variables is important since this may lead to larger savings.

It has been traditionally considered that the CPU is responsible for most of the power being consumed in a server, as we saw in Figure 1.2, and that this power increases linearly with the load. As we could see in that same figure, the power consumed by the CPU is significant, but the power incurred by other elements of the server, like disks and NICs (Network Interface Cards) is not negligible, and have to be taken into account. Moreover, we believe that the assumption that CPU power consumption depends linearly from the load in a server may be too simplistic, especially when the server has multiple cores and may operate at multiple frequencies. In fact, even the way load is expressed has to be carefully defined (e.g., it cannot be defined as a proportion of the maximal computational capacity of the CPU, since this value changes with the operational frequency). Therefore, more complex/complete models for the power consumed by a server are necessary. In order to be consistent, these models have to be based on empirical values. However, we found that there is a lack of empirical work studying servers energy behavior.

The study we present in Chapter 6 tries to partially fill this void by proposing a measurement-based characterization of the energy consumption of a server components with DVFS and multiple cores.

2.4.2 Related Work

There is a large body of work in the field of modeling server energy consumption and its components, both theoretically and empirically. The consumption of servers has been assumed as linear, e.g., by Wang *et al.* [164], Mishra *et al.* [124] or Beloglazov *et al.* [33], who assumed models in which energy consumption mainly depends linearly on CPU utilization. Based on the models, they proposed bin-packing-like algorithms to reduce energy consumption. Other works like the ones from Andrews *et al.* [14] or Irani *et al.* [94] proposed non-linear models, claiming that energy could be saved by running processes at the lowest possible speed.

Moving to the empirical field, we first classify works in two different groups, depending on whether they onsider the effect of frequency in their analysis. We start with works not considering frequency. In this category we find articles proposing models where server components follow a linear behavior, like in [101, 108, 155] or more complex ones, like in [29, 62, 106]. In [108], Liu *et al.* proposed a simple linear model and evaluate different hardware configurations and types of workloads by varying the number of available cores, the available memory, and considering also the contribution of other components such as disks. Vasan *et al.* [155] monitored multiple servers on a datacenter as well as the energy consumption of several of the internal elements of a server. However, they considered that the behavior of this server could be approximated by a model based only on CPU utilization. Similarly, Krishnan *et al.* [101] explored the feasibility of lightweight virtual machine power metering methods and examined the contribution of some of the elements that consume energy in a server like CPU, memory and disks. Their model depends linearly on each of these components. In [62], Economou *et al.* proposed a non-intrusive method for modeling full-system energy consumption by stressing its components with different workloads. Their resulting model is also linear on the utilization of server components. Finally, Lewis *et al.* [106] and Basmasjian *et al.* [29] presented much more complex models which, apart from the contribution of different components of the server, consider extra parameters like temperature and cache misses as well as multiple cores. In particular, Lewis *et al.* [106] reported also an extensive study on the behavior of reading and writing operations in hard disk and solid state drives.

Next we move to the works which also consider frequency in their analysis. Miyoshi *et al.* [125] analyzed the runtime effects of frequency scaling on power and energy. Brihi *et al.* [39] presented an exhaustive study of DVFS using a `cpufrequtils` as we do. Main differences with our work were that they studied four different power management policies under DVFS and centered their study on the relationship between CPU and power utilization. However, they also presented interesting results about disk consumption that match partially our results, showing a flat consumption in reading operations and variations in the writing ones that they attribute to the size of the files being written. Although it was not the main objective of their work, Raghavendra *et al.* [138] performed a per-frequency and core CPU power characterization of two different blade servers. However, they claimed that CPU power depends linearly on its utilization. The main difference with our analysis is that we consider that the load supported by a server increases with the number of active cores and, hence, this load should not be represented in percentage. Gandhi *et al.* [73] published an analysis of global energy consumption versus frequency, based on DVFS and DFS and gave some intuition about the non-linearity of this relation.

Moreover, there are studies that model the energy consumption behavior for clouds and try to balance the load in order to operate the cluster in its most efficient load-power combination. MUSE [49] is one of the first works that consider a resource management architecture for data centers. Its energy efficient approach dynamically assigns jobs to the servers based on the workload (for CPU and disk) and the potential energy consumption. The authors measure the energy consumption of servers and switches involved in the cluster and conclude that at least 29% of the energy can be saved by MUSE for typical web workloads. In [147] the authors proposed a consolidation algorithm that considers the workloads of the servers in the cloud in order to find the least possible energy consumption point. Their study shows that the energy consumption of a server using variable loads for CPU and disks has an optimal operating point. Given the data from the various servers the algorithm can estimate the ideal load distribution among the servers. The authors in [23] modeled the energy consumption of data centers equipment (i.e., servers, storage, switches) for cloud computing based on existing energy consumption measurements or publicly available data sheets for each of the components (CPU, disk, network, switches). The model estimates the energy consumption per bit from the data center to the user and further analyzes the energy consumption for different types of services, i.e., storage, software, processing. However, existing works on clouds lack experimental inputs on energy consumption.

We conclude with some works that also consider frequency but do not model the energy consumption of a server. First of them, the work from Le Sueur *et al.* [104] presented an analysis of the evolution of the effectiveness of DVFS and how it is reduced in the newest and most optimized servers. They show that DVSF might be soon obsoleted by the adoption of ultra low power sleep modes. Ge *et al.* proposed PowerPack [76], a framework that includes a set of toolkits to perform an exhaustive profile of the power utilization of servers and its components. Their analysis is centered in showing the contribution of multicore system to the efficiency of several applications and, hence, no power characterization is presented. Finally, Basmadjian *et al.* [30] published an in deep analysis of the components of a processor and its contribution to the energy consumption of the CPU, shedding some light on the behavior of multicore servers. Some of their conclusions are very relevant to our work, as they show, for instance, that the energy consumption of multiple cores performing parallel computations is not equal to the sum of the power of each of those active cores. Our experiments and model support their findings and shed light on the nature of such effect.

2.5 Power Aware Assignment of Virtual Machines to Physical Machines

2.5.1 Background

The current pace of technology developments, and the continuous change in business requirements, may rapidly yield a given proprietary computational platform obsolete, oversized, or insufficient. Thus, outsourcing has recently become a popular approach to obtain computational services without incurring in amortization costs. Furthermore, in order to attain flexibility, such service is usually virtualized, so that the user may tune the computational platform to its particular needs. Users of such service need not to be aware of the particular implementation, they only need to specify the virtual machine they want to use. This conceptual approach to outsourced computing has been termed cloud computing, in reference to the cloud symbol used as an abstraction of a complex infrastructure in system diagrams. Current examples of cloud computing providers include Amazon Web Services [1], Rackspace [5], and Citrix [3].

Depending on what the specific service provided is, the cloud computing model comes in different flavors, such as infrastructure as a service, platform as a service, storage as a service, etc. In each of these models, the user may choose specific parameters of the computational resources provided. For instance, processing power, memory size, communication bandwidth, etc. Thus, in a cloud-computing service platform, various virtual machines (VM) with user-defined specifications must be implemented by, or assigned to[1], various physical machines (PM)[2]. Furthermore, such a platform must be scalable, allowing to add more PMs, should the business growth require such expansion. In this work, we call this problem the Virtual Machine Assignment (VMA) problem.

The optimization criteria for VMA depends on what the particular objective function sought is. From the previous discussion, it can be seen that, underlying VMA, there is some form of bin-packing problem. However, in VMA the number of PMs (i.e., bins for bin packing) may be increased if needed. Since CPU is generally the dominant power consumer in a server, as shown in Figure 1.2 and as we will show in Chapter 6, VMA is usually carried out according to CPU workloads. With only the static power consumption of servers considered, previous work related to VMA has focused on minimizing the number of active PMs (cf. [32] and the references therein) in order to minimize the total static energy consumption. This is commonly known as VM consolidation [103, 128]. However, despite the static power, the dynamic power consumption of a

[1]The cloud-computing literature use instead the term placement. We choose here the term assignment for consistency with the literature on general assignment problems.

[2]We choose the notation VM and PM for simplicity and consistency, but notice that our study applies to any computational resource assignment problem, as long as the minimization function is the one modeled here.

server, which has been shown to be superlinear on the load of a given computational resource [24, 82], is also significant and cannot be ignored. Since the definition of load is not precise, we use the definition we provide in Chapter 6 and define the load of a server as the amount of active cycles per second a task requires, an absolute metric independent of the operating frequency or the number of cores of a PM. The superlinearity property of the dynamic power consumption is also confirmed by the results that we show in Chapter 6. As a result, when taking into account both parts of power consumption, the use of extra PMs may be more efficient energy-wise than a minimum number of heavily-loaded PMs. This inconsistency with the literature in VM consolidation is supported by the results that will be presented in Chapter 6 and, hence, we claim that the way consolidation has been traditionally performed has to be reconsidered. In this work, we combine both power-consumption factors and explore the most energy-efficient way for VMA. That is, for some parameters $\alpha > 1$ and $b > 0$, we seek to minimize the sum of the α powers of the PMs loads *plus* the fixed cost b of using each PM.

Physical resources are physically constrained. A PMs infrastructure may be strictly constrained in the number of PMs or in the PMs CPU capacity. However, if usage patterns indicate that the PMs will always be loaded well below their capacity, it may be assumed that the capacity is unlimited. Likewise, if the power budget is very big, the number of PMs may be assumed unconstrained for all practical purposes. These cases yield 4 VMA subproblems, depending on whether the capacity and the number of PMs is limited or not. We introduce these parameters denoting the problem as **(*C*,*m*)-VMA**, where C is the PM CPU capacity, m is the maximum number of PMs, and each of these parameters is replaced by a dot if unbounded.

2.5.1.1 Problem Definition

We describe the $(\cdot,\cdot)$-VMA problem now for a better understanding of some of the works presented in the following related work section. Given a set $S = \{s_1, \dots, s_m\}$ of $m > 1$ identical physical machines (PMs) of capacity C; rational numbers μ, α and b, where $\mu > 0$, $\alpha > 1$ and $b > 0$; a set $D = \{d_1, \dots, d_n\}$ of n virtual machines and a function $\ell : D \to \mathbb{R}$ that gives the CPU load each virtual machine incurs[3], we aim to obtain a partition $\pi = \{A_1, \dots, A_m\}$ of D, such that $\ell(A_i) \le C$, for all i. Our objective will be then minimizing the power consumption given by the function

$$P(\pi) = \sum_{i \in [1,m]: A_i \neq \emptyset} \left(\mu \Big(\sum_{d_j \in A_i} \ell(d_j) \Big)^{\alpha} + b \right). \tag{2.2}$$

Let us define the function $f(\cdot)$, such that $f(x) = 0$ if $x = 0$ and $f(x) = \mu x^{\alpha} + b$ otherwise. Then, the objective function is to minimize $P(\pi) = \sum_{i=1}^{m} f(\ell(A_i))$. The parameter μ is used for consistency with the literature.

[3]For convenience, we overload the function $\ell(\cdot)$ to be applied over sets of virtual machines, so that for any set $A \subseteq D, \ell(A) = \sum_{d_j \in A} \ell(d_j)$.

We also define several special cases of the VMA problem, namely (C, m)-VMA, $(C, \cdot)$-VMA, $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA. (C, m)-VMA refers to the case where both the number of available PMs and its capacity are fixed. $(\cdot, \cdot)$-VMA, where $(\cdot)$ denotes unboundedness, refers to the case where both the number of available PMs and its capacity are unbounded (i.e., C is larger than the total load of the VMs that can ever be in the system at any time, or m is larger than the number of VMs that can ever be in the system at any time). $(C, \cdot)$-VMA and $(\cdot, m)$-VMA are the cases where the number of available PMs and their capacity is unbounded, respectively.

2.5.2 Related Work

To the best of our knowledge, previous work on VMA has been only experimental [50, 109, 116, 153] or has focused on different cost functions [11, 32, 48, 54]. First, we provide an overview of previous theoretical work for related assignment problems (storage allocation, scheduling, network design, etc.). The cost functions considered in that work resemble or generalize the power cost function under consideration here. Secondly, we overview related experimental work.

Chandra and Wong [48], and Cody and Coffman [54] study a problem for storage allocation that is a variant of $(\cdot, m)$-VMA with $b = 0$ and $\alpha = 2$. Hence, this problem tries to minimize the sum of the squares of the machine-load vector for a fixed number of machines. They study the offline version of the problem and provide algorithms with constant approximation ratio. A significant leap was taken by Alon *et al.* [11], since they present a PTAS for the problem of minimizing the L_p norm of the load vector, for any $p \geq 1$. This problem has the previous one as special case, and is also a variant of the $(\cdot, m)$-VMA problem when $p = \alpha$ and $b = 0$. Similarly, Alon *et al.* [12] extended this work for a more general set of functions, that include $f(\cdot)$ as defined above. Hence, their results can be directly applied in the $(\cdot, m)$-VMA problem. Later, Epstein *et al.* [66] extended [12] further for the uniformly related machines case. We will use these results in Chapter 7 in the analysis of the offline case of $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA.

Bansal, Chan, and Pruhs minimize arbitrary power functions for speed scaling in job scheduling [24]. The problem is to schedule the execution of n computational jobs on a *single* processor, whose speed may vary within a countable collection of intervals. Each job has a release time, a processing work to be done, a weight characterizing its importance, and its execution can be suspended and restarted later without penalty. A scheduler algorithm must specify, for each time, a job to execute and a speed for the processor. The goal is to minimize the weighted sum of the flow times over all jobs plus the energy consumption, where the flow time of a job is the time elapsed from release to completion and the energy consumption is given by s^α where s is the processor speed and $\alpha > 1$ is some constant. For the online algorithm *shortest remaining processing time first*, the authors prove a $(3 + \epsilon)$ competitive ratio for the objective of total weighted flow plus energy. Whereas for the online algorithm *highest density first (HDF)*, where the density of a job is its weight-to-work ratio, they prove a $(2 + \epsilon)$ competitive ratio for the objective of fractional weighted flow plus energy.

Recently, Im, Moseley, and Pruhs studied online scheduling for general cost functions of the flow time, with the only restriction that such function is non-decreasing [93]. In their model, a collection of jobs, each characterized by a release time, a processing work, and a weight, must be processed by a *single* server whose speed is variable. A job can be suspended and restarted later without penalty. The authors show that HDF is $(2+\epsilon)$-speed $O(1)$-competitive against the optimal algorithm on a unit speed-processor, for all non-decreasing cost functions of the flow time. Furthermore, they also show that this ratio cannot be improved significantly proving impossibility results if the cost function is not uniform among jobs or the speed cannot be significantly increased.

A generalization of the above problem is studied by Gupta, Krishnaswamy, and Pruhs in [82]. The question addressed is how to assign jobs, *possibly fractionally*, to unrelated parallel machines in an online fashion in order to minimize the sum of the α-powers of the machine loads plus the assignment costs. Upon arrival of a job, the algorithm learns the increase on the load and the cost of assigning a unit of such job to a machine. Jobs cannot be suspended and/or reassigned. The authors model a greedy algorithm that assigns a job so that the cost is minimized as solving a mathematical program with constraints arriving online. They show a competitive ratio of α^α with respect to the solution of the dual program which is a lower bound for the optimal. They also show how to adapt the algorithm to integral assignments with a $O(\alpha)^\alpha$ competitive ratio, which applies directly to our $(\cdot, m)$-VMA problem. References to previous work on the particular case of minimizing energy with deadlines can be found in this paper.

Similar cost functions have been considered for the minimum cost network-design problem. In this problem, packets have to be routed through a (possibly multihop) network of speed scalable routers. There is a cost associated to assigning a packet to a link and to the speed or load of the router. The goal is to route all packets minimizing the aggregated cost. In [13] and [14] the authors show offline algorithms for this problem with undirected graph and homogeneous link cost functions that achieve polynomial and poly-logarithmic approximation, respectively. The cost function is the α-th power of the link load plus a link assignment cost, for any constant $\alpha > 1$. The same problem and cost function is studied in [82]. Bansal *et al.* [25] study a minimum-cost virtual circuit multicast routing problem with speed scalable links. They give a polynomial-time $O(\alpha)$-approximation offline algorithm and a polylog-competitive online algorithm, both for the case with homogeneous power functions. They also show that the problem is APX-hard in the case with heterogeneous power functions and there is no polylog-approximation when the graph is directed. Recently, Antoniadis *et al.* [17] improved the results by providing a simple combinatorial algorithm that is $O(\log^\alpha n)$-approximate, from which we can construct an $\widetilde{O}(\log^{3\alpha+1} n)$-competitive online algorithm. The $(\cdot, m)$-VMA problem can be seen as a especial case of the problem considered in these papers in which there are only two nodes, source and destination, and m parallel links connecting them.

To the best of our knowledge, the problem of minimizing the power consumption (given in Eq.2.2) with capacity constraints (i.e., the (C, m)-VMA and $(C, \cdot)$-VMA problems) has received very limited attention, in the realm of both VMA and network design, although the approaches in [14] and [25] are related to or based on the solutions for the capacitated network-design problem [47].

The experimental work related to VMA is vast and its detailed overview is out of the scope of this paper. Some of this work does not minimize energy [45, 110, 123] or it applies to a model different than ours (VM migration [130, 147], knowledge of future load [112, 147], feasibility of allocation [32], multilevel architecture [96, 116, 130], interconnected VMs [37], etc.). On the other hand, some of the experimental work where minimization of energy is evaluated focus on a more restrictive cost function [96, 159, 168].

In [96], for an energy cost model that is linear, the authors evaluate experimentally the allocation of VMs to clusters following 9 placement policies, some of them included in popular cloud platforms [68, 135]. Namely, Round Robin, Striping, Packing, Load Balancing (free CPU count), Load Balancing (free CPU ratio), Watts per Core, Cost per Core. We adapt 5 of these policies (defined later in Chapter 7) to our model and cost function for the purpose of simulations.

In [147], the authors focus on an energy-efficient VM placement problem with two requirements: CPU and disk. These requirements are assumed to change dynamically and the goal is to consolidate loads among servers, possibly using migration at no cost. In our model VMs assignment is based on a CPU requirement that does not change and migration is not allowed. Should any other resource be the dominating energy cost, the same results apply for that requirement. Also, if loads change and migration is free, an offline algorithm can be used each time that a load changes or a new VM arrives. In [147] it is shown experimentally that energy-efficient VMA does not merely reduce to a packing problem. That is, to minimize the number of PMs used even if their load is close to their maximum capacity. For our model, we show here that the optimal load of a given server is a function only of the fixed cost of being active (b) and the exponential rate of power increase on the load (α). That is, the optimal load is not related to the maximum capacity of a PM.

Part II

Structural Issues

Chapter 3

Bisection Width and Bisection Bandwidth of Product Networks

3.1 Overview

In this chapter we study different families of product networks and how to compute their bisection width and bandwidth. In particular, we propose two theorems that will allow us to derive upper and lower bounds on the bisection width of a product network as a function of some simple parameters of its factor graphs.

We prove that this method allows to compute the exact bisection width of several traditional interconnection networks. Concretely, we study the families of product graphs derived from the combination of paths and complete binary trees and the product graphs derived from the combination of rings and extended trees. Among these results, the bisection width of the multidimensional torus stands out as the most remarkable one, given that, as mentioned in Chapter 1, it has been a long standing problem, unsolved for more than 20 years. To obtain the bisection bandwidth of these networks, we assume that every edge removed by the bisection width is in fact a duplex link with bandwidth of T in each direction. This directly implies that for any of these networks G, the bisection bandwidth is computed as $BBW(G) = 2T \cdot BW(G)$

Apart from traditional interconnection networks, we also target newly proposed topologies for data centers. In particular, we show how the bisection width of BCube can be also approximated by using this method. However, there is an important difference between this and the previous topologies. While the traditional networks only had one kind of nodes, new topologies such as BCube have, at least, two different kinds, in this case, servers and switches. This fact leads to an essential difference regarding the traditional ones. Now, edges do not connect nodes directly, and, hence, the direct relation between bisection width and bisection bandwidth does not hold anymore.

A BCube is the Cartesian product of factors networks formed by k nodes connected via a k-port switch. In networks with switches like this one, the switching capacity s of the switch

Table 3.1: Bisection bandwidth of different product networks

Product graph	**Factor graphs**		$\beta(G)$	$CC(G)$	**Bisection bandwidth**
Torus	Ring		$1/8$	2	$4T \cdot \Psi(\alpha)$
Product of extended CBT	XTs		$1/8$	2	$4T \cdot \Psi(\alpha)$
Product of extended CBT & rings	Rings & XTs		$1/8$	2	$4T \cdot \Psi(\alpha)$
Mesh connected trees	CBT		$1/4$	1	$2T \cdot \Psi(\alpha)$
Product of CBT and paths	Paths & CBTs		$1/4$	1	$2T \cdot \Psi(\alpha)$
BCube	Model A	even	$\frac{k-1}{k^2}$	$\frac{k}{2}$	$2T\frac{k^{d+1}}{4(k-1)} \leq BBW(BCA_k^{(d)}) \leq 2T\frac{k^d}{2}$
		odd	$\frac{1}{k+1}$	$\frac{k-1}{2}$	$2T\frac{k+1}{4}\frac{k^d-1}{k-1} \leq BBW(BCA_k^{(d)}) \leq 2T\frac{k^d-1}{2}$
	Model B	even	$\frac{k-1}{2k}$	1	$s\frac{k^d}{2(k-1)} \leq BBW(BCB_k^{(d)}) \leq s\frac{k^d-1}{k-1}$
		odd	$\frac{k}{2(k+1)}$	1	$s\frac{k+1}{2k}\frac{k^d-1}{k-1} \leq BBW(BCB_k^{(d)}) \leq s\frac{k^d-1}{k-1}$

comes into play as well. Since the bisection bandwidth is the parameter of interest in datacenters, we derive bounds on its value for two cases: when the bottleneck for the bisection bandwidth is at the links (Model A), and when it is at the switches (Model B).

Table 3.1 summarizes the results derived for the bisection bandwidth obtained for the different parallel topologies and for BCube. As can be seen, for the former the values obtained are exact, while for the latter the upper and lower bounds found do no match exactly. However, they differ by less than a factor of two. The value $\Psi(\alpha)$ introduced in Table 3.1 denotes the bisection width of the multidimensional array. The value of the bisection width of the array is $\Psi(\alpha) = BW(A^{(d)}_{k_1,k_2,\ldots,k_d}) = \sum_{i=1}^{\alpha} C_i = \sum_{i=1}^{\alpha}\prod_{j=i+1}^{d} k_j$. For readability, since the value of the bisection width of the array will appear frequently, we will use the notation $\Psi(\alpha)$, throughout Chapter 3.

RoadMap The rest of this chapter is organized as follows. Section 3.2 presents some basic definitions used in the rest of the chapter. In Section 3.3 we provide the general results to derive bounds on the bisection bandwidth of product networks. Section 3.4 and Section 3.5 present our results for the bisection bandwidth of some classical parallel topologies. Bounds on the bisection bandwidth of the BCube network are presented in Section 3.6. Finally, in Section 3.7 we present our conclusions and some open problems.

3.2 Definitions

3.2.1 Graphs and Bisections

In this section we present definitions and notation that will be used along the text. Given a graph[1] G, we denote its sets of vertices and edges as $V(G)$ and $E(G)$, respectively. In some

[1]Unless otherwise stated we will use the terms graph and network indistinctly throughout this chapter.

cases, when it is clear from the context, only V or E will be used, omitting the graph G. Unless otherwise stated, the graphs considered are undirected.

Given a graph G with n nodes, we use $S(G)$ to denote a subset of $V(G)$ such that $|S(G)| \leq \frac{n}{2}$. We also use $\partial^G S(G)$ to denote the set of edges connecting $S(G)$ and $V(G) \setminus S(G)$. Formally, $\partial^G S(G) = \{(u, v) \in E(G) : u \in S(G), v \in G \setminus S(G)\}$. The graph G may be omitted from this notation when it is clear from the context.

The main object of the work presented in this chapter is to calculate the bisection width and bisection bandwidth of different product networks. These bisections can be defined as follows.

Definition 1. *The* bisection width *of an n-node graph G, denoted $BW(G)$, is the smallest number of edges that have to be removed from G to partition it in two halves. Formally, $BW(G) = \min_{S:|S|=\lfloor \frac{n}{2} \rfloor} |\partial^G S|$.*

Definition 2. *The* bisection bandwidth *of a network G, denoted $BBW(G)$, is the minimal amount of data per time unit that can be transferred between any two halves of the network when its links are transmitting at full speed.*

As mentioned above, unless otherwise stated we assume that all the links in a network G are duplex and have the same capacity T in each direction. Then, we can generally assume that the relation between the bisection bandwidth and the bisection width is $BBW(G) = 2T \cdot BW(G)$.

3.2.2 Factor and Product Graphs

We first define the Cartesian product of graphs.

Definition 3. *The d-*dimensional Cartesian product *of graphs $G_1, G_2, ..., G_d$, denoted by $G_1 \times G_2 \times \cdots \times G_d$, is the graph with vertex set $V(G_1) \times V(G_2) \times \cdots \times V(G_d)$, in which vertices $(u_1, ..., u_i, ..., u_d)$ and $(v_1, ..., v_i, ..., v_d)$ are adjacent if and only if $(u_i, v_i) \in E(G_i)$ and $u_j = v_j$ for all $j \neq i$.*

The graphs $G_1, G_2, ..., G_d$ are called the *factors* of $G_1 \times G_2 \times \cdots \times G_d$. Observe that $G_1 \times G_2 \times \cdots \times G_d$ contains $\prod_{j \neq i} |V(G_j)|$ disjoint copies of G_i, which form dimension i. We now define some of the basic factor graphs that will be considered.

Definition 4. *The* path *of k vertices, denoted by P_k, is a graph such that $V(P_k) = \{0, 1, \ldots, k-1\}$ and where $E(P_k) = \{(i, i+1) : i \in [0, k-2]\}$.*

Definition 5. *The* complete graph *(a.k.a. the clique) of k vertices, denoted by K_k, is a graph such that $V(K_k) = \{0, 1, \ldots, k-1\}$ and where $E(K_k) = \{(i, j) : (j \neq i) \wedge (i, j \in V(K_k))\}$.*

Definition 6. *The* r-complete graph *of k vertices denoted by rK_k, is a graph such that $V(rK_k) = \{0, 1, \ldots, k-1\}$ and where $E(rK_k)$ is a multiset such that each pair of vertices $i, j \in V(rK_k)$ is connected with r parallel edges.*

Using these and other graphs as factors, we will define, across the text, different d-dimensional Cartesian product graphs. For convenience, for these graphs we will use the general notation $G^{(d)}_{k_1,\ldots,k_d}$, where G is the name of the graph, the superscript (d) means that it is a d-dimensional graph, and $k_1,\ldots k_d$ are the number of vertices in each dimension. (Superscript and subscripts may be omitted when clear from the context.) It will always hold that $k_1 \geq k_2 \geq \ldots \geq k_d$, i.e., the factor graphs are sorted by decreasing number of vertices. We will often use n to denote the number of nodes in the graph $G^{(d)}_{k_1,\ldots,k_d}$, i.e., $n = k_1k_2\cdots k_d$, and we will always use α to denote the index of the lowest dimension with an even number of vertices (if there is no such dimension, $\alpha = d$, where d is the index of the lowest dimension). According to this notation we will present different d-dimensional product graphs as follows.

Definition 7. *The* d-dimensional array*, denoted by $A^{(d)}_{k_1,\ldots,k_d}$, is the Cartesian product of d paths of $k_1,\ldots,k_d$ vertices, respectively. I.e., $A^{(d)}_{k_1,\ldots,k_d} = P_{k_1} \times P_{k_2} \times \cdots \times P_{k_d}$.*

Definition 8. *The* d-dimensional r-Hamming graph*, denoted by $rH^{(d)}_{k_1,\ldots,k_d}$, is the Cartesian product of d r-complete graphs of $k_1,\ldots,k_d$ nodes, respectively. I.e., $rH^{(d)}_{k_1,\ldots,k_d} = rK_{k_1} \times rK_{k_2} \times \cdots \times rK_{k_d}$.*

Observe that the *Hamming graph* [20] is the particular case of the r-Hamming graph, with $r = 1$. For brevity, we use $H^{(d)}_{k_1,\ldots,k_d}$ instead of $1H^{(d)}_{k_1,\ldots,k_d}$, to denote the Hamming graph.

3.2.3 Boundaries and Partitions

We define now the dimension-normalized boundary [21].

Definition 9. *Let $G^{(d)}_{k_1,\ldots,k_d}$ be a d-dimensional product graph and $S(G)$ a subset of $V(G)$. Then, the* dimension-normalized boundary *of $S(G)$, denoted by $B_G(S)$, is defined as*

$$B_G(S) = \frac{|\partial^G_1 S|}{\sigma_1} + \frac{|\partial^G_2 S|}{\sigma_2} + \ldots + \frac{|\partial^G_d S|}{\sigma_d}, \tag{3.1}$$

where, for each $i \in [1,d]$, ∂^G_i is the subset of the edges in the boundary ∂^G that belong to dimension i, and σ_i is defined as

$$\sigma_i = \begin{cases} k_i^2 & \text{if } k_i \text{ is even} \\ k_i^2 - 1 & \text{if } k_i \text{ is odd.} \end{cases} \tag{3.2}$$

Observation 1. *For $rH^{(d)}_{k_1,\ldots,k_d}$, any subset S of nodes, and any dimension i, it holds that $|\partial^{rH}_i S| = r \cdot |\partial^H_i S|$. Hence,*

$$B_{rH}(S) = \frac{|\partial^{rH}_1 S|}{\sigma_1} + \cdots + \frac{|\partial^{rH}_d S|}{\sigma_d} = r\left(\frac{|\partial^H_1 S|}{\sigma_1} + \cdots + \frac{|\partial^H_d S|}{\sigma_d}\right) = rB_H(S).$$

Table 3.2: Basic notation in Chapter 3.

Notation	
$(B)BW(G)$	Bisection (Band)Witdh of graph G
n	Number of nodes in graph G
k_i	Number of nodes in dimension i
d	Dimension index
$G^{(d)}_{k_1,\ldots,k_d}$	Graph G with d dimensions of sizes $k_1, k_2, \ldots, k_d$
α	Lowest index of an even k_i
$\Psi(\alpha)$	Bisection Width of a d-dimensional array
$\partial^g S(G)$	Edges connecting $S(G)$ and $V(G) \setminus S(G)$
$B_g(S)$	Dimension normalized boundary of $S(G)$
$CC(G)$	Central Cut of graph G
$\beta_r(G)$	Normalized congestion of graph G of multiplicity r
$m_r(G)$	Congestion of graph G with multiplicity r
$\mathcal{E}$	Set of all possible embeddings M_r of rK_n onto G
T	Links capacity
s	Switching capacity

Let us define the lexicographic-order. Consider graph $H^{(d)}_{k_1,\ldots,k_d}$, we say that vertex $x = (x_1, x_2, \ldots, x_d)$ precedes vertex $y = (y_1, y_2, \ldots, y_d)$ in *lexicographic-order* if there exists an index $i \in [1, d]$ such that $x_i < y_i$ and $x_j = y_j$ for all $j < i$. Azizoglu and Egecioglu [20] proved the following result.

Theorem 1 ([20]). *Consider a d-dimensional Hamming graph $H^{(d)}_{k_1,\ldots,k_d}$, with $k_1 \geq k_2 \geq \cdots \geq k_d$. Let S be any subset of $V(H)$ and $\bar{S}$ the set of first $|S|$ vertices of H in lexicographic-order*[2]*, then $B_H(\bar{S}) \leq B_H(S)$.*

Table 3.2 summarizes the basic notation used in this chapter.

3.3 Bounds on the BW of Product Graphs

In this section we present general bounds on the bisection width of product graphs as well as two important parameters, the normalized congestion and the central cut, which are used to obtain them. These bounds will be used in the upcoming sections to find the bisection width of several instances of product graphs.

3.3.1 Lower Bound

We start by defining the normalized congestion of a graph. Let G be a graph with n nodes. Then, an *embedding* of the graph rK_n onto G is a mapping of the edges of rK_n into paths in G. We define the *congestion of G with multiplicity r*, denoted by $m_r(G)$, as the minimum (over

[2]Observe that we have reversed the ordering of dimensions with respect to the original theorem of [20].

all such embeddings) of the maximum number of embedded paths that contain an edge from G. To formally define this concept, we first define *the congestion of an edge* $e \in E(G)$ under an embedding M_r of rK_n onto G as $c_{M_r}(e) = |\{e' \in E(rK_n) : e \in M_r(e')\}|$. (Observe that $M_r(e') \subseteq E(G)$ is a path in G.) Then, the congestion $m_r(G)$ is

$$m_r(G) = \min_{M_r \in \mathcal{E}} \max_{e \in E(G)} \{c_{M_r}(e)\}, \tag{3.3}$$

where $\mathcal{E}$ is the set of all possible embeddings of rK_n onto G. Then, using Eqs. (3.3) and (3.2), we define the *normalized congestion* with multiplicity r of G as $\beta_r(G) = m_r(G)/\sigma_n$. Having defined the normalized congestion, we proceed to extend Theorem 1 to r-Hamming graphs.

Theorem 2. *Consider a d-dimensional r-Hamming graph $rH^{(d)}$. Let S be any vertex subset of $V(rH^{(d)})$ and $\bar{S}$ the set of first $|S|$ vertices of $rH^{(d)}$ in lexicographic order, then $B_{rH}(\bar{S}) \leq B_{rH}(S)$.*

Proof: We prove the theorem by contradiction. Assume that there is a set of vertices $X \neq \bar{S}$ such that $|X| = |\bar{S}|$ and $B_{rH}(\bar{S}) > B_{rH}(X)$. Then, applying Observation 1 to both X and $\bar{S}$, we obtain that $B_H(\bar{S}) = B_{rH}(\bar{S})/r > B_{rH}(X)/r = B_H(X)$, which contradicts Theorem 1 and proves the theorem. ■

We now present the following lemma.

Lemma 1. *Let $\bar{S}$ be a subset of the vertices of graph $rH^{(d)}_{k_1,k_2,\ldots,k_d}$, such that $\bar{S}$ are the first $\lfloor \frac{n}{2} \rfloor$ vertices of rH in lexicographic order, and n is the number of vertices of rH. Then, the dimension-normalized boundary of $\bar{S}$ is $B_{rH}(\bar{S}) = r\Psi(\alpha)/4$.*

Proof: We will derive first the value of $B_H(\bar{S})$, and then use Observation 1 to prove the claim. It was shown in [21], that $\partial_i^H \bar{S} = \emptyset$ for all $i > \alpha$.[3] The number of edges in each dimension $i \in [1, \alpha]$ on the boundary of $\bar{S}$ in H is

$$|\partial_i^H \bar{S}| = \begin{cases} \frac{k_i}{2}(\prod_{j=i+1}^{d} k_j)\frac{k_i}{2} & \text{if } k_i \text{ is even} \\ \frac{k_i-1}{2}(\prod_{j=i+1}^{d} k_j)\frac{k_i+1}{2} & \text{if } k_i \text{ is odd.} \end{cases} \tag{3.4}$$

Then, from the definition of $B_H(\bar{S})$, we obtain that

$$\begin{aligned} B_H(\bar{S}) &= \frac{\frac{k_1-1}{2}(\prod_{j=2}^{d} k_j)\frac{k_1+1}{2}}{k_1^2 - 1} + \frac{\frac{k_2-1}{2}(\prod_{j=3}^{d} k_j)\frac{k_2+1}{2}}{k_2^2 - 1} \frac{\frac{k_\alpha}{2}(\prod_{j=\alpha+1}^{d} k_j)\frac{k_\alpha}{2}}{k_\alpha^2} \\ &= \frac{\prod_{j=2}^{d} k_j}{4} + \frac{\prod_{j=3}^{d} k_j}{4} + \cdots + \frac{\prod_{j=\alpha+1}^{d} k_j}{4} = \frac{\sum_{i=1}^{\alpha} C_i}{4} = \frac{\Psi(\alpha)}{4}. \end{aligned}$$

Finally, from Observation 1, we derive $B_{rH}(\bar{S}) = rB_H(\bar{S}) = r\Psi(\alpha)/4$. ■

Using Definition 3, Lemma 1, and Eq. (3.3), we obtain the following theorem.

[3]Observe that they use reverse lexicographic order and sort dimensions in the opposite order we do.

Theorem 3. *Let $G = G_1 \times \ldots \times G_d$, where $|V(G_i)| = k_i$ and $k_1 \geq k_2 \geq \ldots \geq k_d$. Let $\beta_r(G_i)$ be the normalized congestion with multiplicity r of G_i (for any r), for all $i \in [1, d]$. Consider any subset $S \subset V(G)$ and the subset $\bar{S}$ which contains the first $|S|$ vertices of G, in lexicographic order. Then, $B_{rH}(\bar{S}) \leq \sum_{i=1}^{d} \beta_r(G_i)|\partial_i^G S|$.*

Proof: First, observe that, for any $S_i \subset V(G_i)$, $|\partial^{rK_{k_i}} S_i| \leq m_r(G_i) \cdot |\partial^{G_i} S_i|$. Then, for $S \subset V(G)$ as defined, $|\partial_i^{rH} S| \leq m_r(G_i) \cdot |\partial_i^G S|$. Finally, using Theorem 2, we can state that

$$\begin{aligned} B_{rH}(\bar{S}) &\leq B_{rH}(S) \\ &\leq m_r(G_1)\frac{|\partial_1^G S|}{\sigma_1} + \cdots + m_r(G_d)\frac{|\partial_d^G S|}{\sigma_d} \\ &= \beta_r(G_1)|\partial_1^G S| + \cdots + \beta_r(G_i)|\partial_d^G S|. \end{aligned}$$

■

From this theorem, we derive a corollary for the case of $|S| = \lfloor \frac{n}{2} \rfloor$.

Corollary 1. *Let $G = G_1 \times \ldots \times G_d$, where $|V(G_i)| = k_i$ and $k_1 \geq k_2 \geq \ldots \geq k_d$. Let $\beta_r(G_i)$ be the normalized congestion with multiplicity r of G_i (for any r), for $i \in [1, d]$. Consider any subset $S \subset V(G)$ such that $|S| = \lfloor \frac{|V(G)|}{2} \rfloor$. Then $r\Psi(\alpha)/4 \leq \sum_{i=1}^{d} \beta_r(G_i)|\partial_i^G S|$.*

Corollary 2. *Let $G = G_1 \times \ldots \times G_d$, where $|V(G_i)| = k_i$ and $k_1 \geq k_2 \geq \ldots \geq k_d$. Let $\beta_r(G_i) = \beta$ be the normalized congestion with multiplicity r of G_i (for any r), for $i \in [1, d]$. Consider any subset $S \subset V(G)$ such that $|S| = \lfloor \frac{|V(G)|}{2} \rfloor$. Then $r\Psi(\alpha)/4\beta \leq BW(G)$.*

3.3.2 Upper Bound

Having proved the lower bound on the bisection width, we follow with the upper bound. We define first the central cut of a graph G.

Consider a graph G with n nodes, and a partition of $V(G)$ into three sets S^-, S^+, and S, such that $|S^-| = |S^+| = \lfloor \frac{n}{2} \rfloor$ (observe that if n is even then $S = \emptyset$, otherwise $|S| = 1$). Then, the *central cut of* G, denoted by $CC(G)$, is

$$\min_{\{S^-, S^+, S\}} \max\{|\partial^G S^-|, |\partial^G S^+|\}.$$

Observe that, for even n, the central cut is the bisection width. Now, we use the definition of central cut in the following theorem.

Theorem 4. *Let $G = G_1 \times \ldots \times G_d$, where $|V(G_i)| = k_i$ and $k_1 \geq k_2 \geq \ldots \geq k_d$. Then, $BW(G) \leq \max_i \{CC(G_i)\} \cdot \Psi(\alpha)$.*

Proof: It was shown in [21] how to bisect $A^{(d)}$ by cutting exactly $BW(A^{(d)}) = \Psi(\alpha)$ links. Furthermore, this bisection satisfies that, if the paths P_{k_i} in dimension i are cut, each of them can be partitioned into subpaths P^+ and P^- of size $\lfloor \frac{k_i}{2} \rfloor$ (connected by a link if k_i is even or by a node with links to both if k_i is odd) so that the cut separates P^+ or P^- from the rest of the path. Each path is then cut by removing one link. We map the sets S^+ and S^+ of the partition that gives the central cut of G_i to P^+ and P^-, respectively. Then, any cut of a paths P_{k_i} in dimension i becomes a cut of G_i with at most $CC(G_i)$ links removed.

Then, if S is the subset of $V(G)$ that ends at one side of the bisection described above, we have that $|\partial_i^G S|/CC(G_i) \leq |\partial_i^{A^{(d)}} S|$, which also holds if the paths in dimension i are not cut. Applying this to all dimensions, we obtain

$$\frac{|\partial_1^G S|}{CC(G_1)} + \cdots + \frac{|\partial_d^G S|}{CC(G_d)} \leq BW(A^{(d)}) = \Psi(\alpha). \tag{3.5}$$

This yields, $BW(G) \leq |\partial_1^G S| + \cdots + |\partial_d^G S| \leq \max_i \{CC(G_i)\} \cdot \Psi(\alpha)$, proving Theorem 4. ■

3.4 BW of Products of CBTs and Paths

In this section we will obtain the bisection bandwidth of product graphs which result from the Cartesian product of paths and Complete Binary Trees (CBT). We will present, first, the different factor graphs we are using and the product graphs we are bisecting, then, we will compute the congestion and central cut of these factor graphs and, finally, calculate the bisection width of these product graphs.

3.4.1 Factor and Product Graphs

In this section we will work with paths, which were defined in Section 3.2, and CBTs, which is defined now.

Definition 10. *The* complete binary tree *of k vertices, denoted by CBT_k, is a graph such that $V(CBT_k) = \{1, 2, \ldots, k\}$, with $k = 2^j - 1$ (j is the number of levels of the tree), and where $E(CBT_k) = \{(i, j) : ((j = 2i) \vee (j = 2i + 1)) \wedge (i \in [1, 2^{j-1} - 1])\}$.*

Combining these factor graphs through the Cartesian product, we obtain the product networks that we define below.

Definition 11. *A d-*dimensional mesh-connected trees and paths*, denoted by $MCTP^{(d)}_{k_1,k_2,\ldots,k_d}$, is the Cartesian product of d graphs of $k_1, k_2, \ldots, k_d$ vertices, respectively, where each factor graph is a complete binary tree or a path. I.e., $MCTP^{(d)}_{k_1,k_2,\ldots,k_d} = G_{k_1} \times G_{k_2} \times \cdots \times G_{k_d}$, where either $G_{k_i} = CBT_{k_i}$ or $G_{k_i} = P_{k_i}$.*

We also define the *d-dimensional mesh-connected trees* [65], denoted by $MCT^{(d)}_{k_1,k_2,\ldots,k_d}$ as the graph $MCTP^{(d)}_{k_1,k_2,\ldots,k_d}$ in which all the factor graphs are complete binary trees. (Observe that the array is also the special case of $MCTP^{(d)}_{k_1,k_2,\ldots,k_d}$ in which all the factor graphs are paths.)

3.4.2 Congestion and Central Cut of Paths and CBTs

The bisection widths of the aforementioned product graphs can be calculated using the bounds defined in Section 3.3. To do so, we need to compute first the values of the normalized congestion and central cut of their factor graphs, it is, of a path and of a CBT.

We start by computing the congestion of a path and of a CBT and, then, their central cuts. We present the following lemma.

Lemma 2. *The congestion of P_k with multiplicity r, denoted $m_r(P_k)$, has two possible values, depending on whether the number of vertices k is even or odd, as follows,*

$$m_r(P_k) = \begin{cases} r\frac{k^2}{4} & \textit{if } k \textit{ is even} \\ r\frac{k^2-1}{4} & \textit{if } k \textit{ is odd} \end{cases} \tag{3.6}$$

Proof: This proof is illustrated in Figure 3.1 where it can be seen that there are two possible cases, depending on whether k is even or odd. The congestion $m_r(P_k)$ is defined as the minimum congestion over all embeddings of rK_k onto P_k. As there is only one possible path between every pair of vertices, the congestion of an edge will always be the same for any embedding M_r of rK_k into P_k. Let M_r be an embedding of rK_k onto P_k. Then,

$$m_r(P_k) = \min_{M\in\mathcal{E}} \max_{e\in E(P_k)} \{c_M(e)\} = \max_{e\in P_k}\{c_{M_r}(e)\}. \tag{3.7}$$

If we fix $e = (i, i+1) \in E(P_k)$, $i \in [0, k-1]$, the congestion of e follows the equation

$$c_{M_r}(e) = r(i+1)(k-i-1). \tag{3.8}$$

The value of i that maximizes $c_{M_r}(e)$ is $i = \frac{k}{2} - 1$. As k is an integer, depending on whether k is even or odd, $\frac{k}{2}$ will be exact or not. Hence, we consider two possible cases, $i = k/2 - 1$ when k is even or $i = (k-1)/2 - 1$ when k is odd. Using these values in Eq. (3.8) leads to the final result

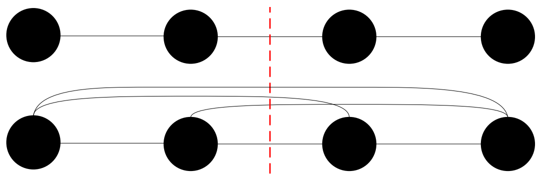
$$m_r(P_k) = \begin{cases} r\frac{k^2}{4} & \text{if } k \text{ is even} \\ r\frac{k^2-1}{4} & \text{if } k \text{ is odd} \end{cases}$$

■

Corollary 3. *The normalized congestion of a path is $\beta_r(P_k) = \frac{r}{4}$.*

The value of the congestion of a CBT is exactly the same as the congestion of a path with an odd number of nodes. CBT share with paths the property of having only one possible routing between two nodes. As can be seen in Figures 3.1 and 3.2, the possible cuts are similar. We present Lemma 3 for the congestion of a CBT.

Lemma 3. *The congestion of CBT_k with multiplicity r is $m_r(CBT_k) = r(k^2-1)/4$.*


(a) The 4-vertex path and clique.

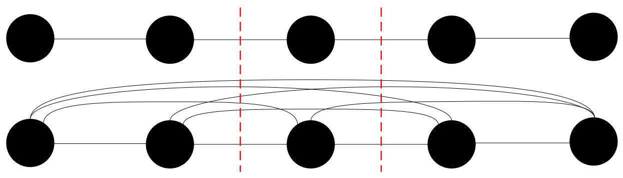

(b) The 5-vertex path and clique.

Figure 3.1: Paths and possible cuts.

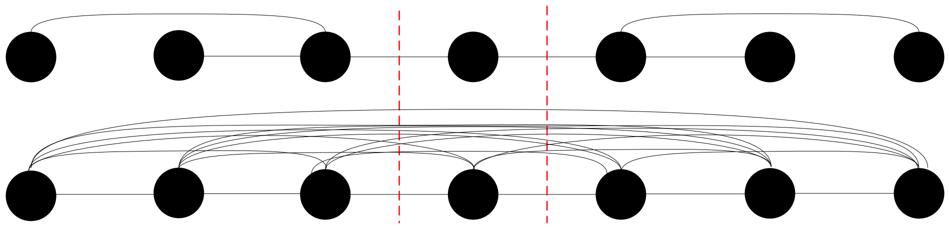


Figure 3.2: The 7-vertex complete binary tree and the 7-vertex clique, with their possible cuts.

Proof: Let CBT_{2^j-1} be a complete binary tree of j levels with $k = 2^j - 1$ nodes. Whichever edge we cut results on two parts, one of them being another complete binary tree, let us call it A and assume it has $l < j$ levels; and the other being the rest of the previous complete binary tree, let us call it B. The number of nodes in A will be $2^l - 1$ while the number of nodes in B will be $k - 2^l + 1$. For any embedding M of rK_k into CBT_k, the congestion of any edge e follows the equation $c_{M_r}(e) = r(2^l-1)(k-2^l+1)$. The value of l which maximizes the equation is $l = j-1$, which is equivalent to cut one of the links of the root. This divides the tree into subgraphs of sizes $\frac{k+1}{2}$ and $\frac{k-1}{2}$. Then, the final value for congestion will be $m_r(CBT_k) = r(k^2 - 1)/4$. ■

Corollary 4. *The normalized congestion of a CBT is* $\beta_r(CBT_k) = r/4$.

The value of thecentral cut of both the path and CBT can also be easily deduced from Figures 3.1 and 3.2, being $CC(P_k) = CC(CBT_k) = 1$.

3.4.3 Bounds on the BW of Products of CBTs and Paths

Having presented both the congestion and the central cut of the possible factor graphs, we can compute now the lower and upper bound on the bisection width of a product of CBTs and paths. We will start by the lower bound on the bisection width.

Lemma 4. *The bisection width of a d-dimensional mesh-connected trees and paths, $MCTP^{(d)}$, is lower bounded by $\Psi(\alpha)$.*

Proof: As we can see in Corollaries 3 and 4, the normalized congestion of both factor graphs is the same value $r/4$. Then, we can apply Corollary 2, so $r\Psi(\alpha)(4r/4) \leq BW(MCTP^{(d)})$, which yields, $BW(MCTP^{(d)}) \geq \Psi(\alpha)$. ■

We follow now by presenting an upper bound on the bisection width of d-dimensional mesh-connected trees and paths.

Lemma 5. *The bisection width of a d-dimensional mesh-connected trees and paths, $MCTP^{(d)}$, is upper bounded by $\Psi(\alpha)$.*

Proof: Obviously, as this graph can also be embedded into a d-dimensional array, we can use Theorem 4. We know that the central cut of both CBTs and paths is 1 independently of their sizes or number of levels, and hence also $\max_i\{CC(G_{k_i})\} = 1$ (where G_{k_i} is either a CBT or a path). Then, $BW(MCTP^{(d)}) \leq \Psi(\alpha)$. ■

From the results obtained from Lemma 4 and Lemma 5 the proof of Theorem 5 follows.

Theorem 5. *The bisection width of a d-dimensional mesh-connected trees and paths $MCTP^{(d)}_{k_1,k_2,\ldots,k_d}$ is $\Psi(\alpha)$.*

We can also present the following corollary for the particular case of the d-dimensional mesh-connected trees $MCT^{(d)}_{k_1,k_2,\ldots,k_d}$.

Corollary 5. *The bisection width of the d-dimensional mesh-connected trees $MCT^{(d)}_{k_1,k_2,\ldots,k_d}$ is $BW(MCT^{(d)}) = \Psi(d)$.*

3.5 Products of Rings and Extended Trees

In this section we will obtain a result for the bisection bandwidth of the product graphs which result from the Cartesian product of rings and extended complete binary trees.

3.5.1 Factor and Product Graphs

The factor graphs which are going to be used in this section are rings and XTs. We define them below.

Definition 12. *The* ring *of k vertices, denoted by R_k, is a graph such that $V(R_k) = \{0, 1, \ldots, k-1\}$ and where $E(R_k) = \{(i, (i+1) \mod k) : i \in V(R_k)\}$.*

Definition 13. *The* extended complete binary tree *(a.k.a. XT) of k vertices, denoted by X_k, is a complete binary tree in which the leaves are connected as a path. More formally, $V(X_k) = V(CBT_k)$ and $E(X_k) = E(CBT_k) \cup \{(i, i+1) : i \in [2^{j-1}, 2^j - 2]\}$.*

Combining these graphs as factor graphs in a Cartesian product we can obtain the following product graphs:

Definition 14. *A* d-dimensional mesh-connected extended trees and rings, *denoted by* $MCXR^{(d)}_{k_1,k_2,\dots,k_d}$*, is the Cartesian product of* d *graphs of* $k_1, k_2, \dots, k_d$ *vertices, respectively, where each factor graph is a extended complete binary tree or a ring. I.e.,* $MCXR^{(d)}_{k_1,k_2,\dots,k_d} = G_{k_1} \times G_{k_2} \times \cdots \times G_{k_d}$*, where either* $G_{k_i} = X_{k_i}$ *or* $G_{k_i} = R_{k_i}$*.*

Definition 15. *The* d-dimensional torus, *denoted by* $T^{(d)}_{k_1,k_2,\dots,k_d}$*, is the Cartesian product of* d *rings of* $k_1, k_2, \dots, k_d$ *vertices, respectively. I.e.,* $T^{(d)}_{k_1,k_2,\dots,k_d} = R_{k_1} \times R_{k_2} \times \cdots \times R_{k_d}$*.*

And, as happened in Section 3.4 with $MCT^{(d)}$, we also define the *d-dimensional mesh-connected extended trees*, denoted by $MCX^{(d)}_{k_1,k_2,\dots,k_d}$, a special case of $MCXR^{(d)}_{k_1,k_2,\dots,k_d}$ in which all factor graphs are XTs. (The torus is the special case of $MCXR^{(d)}_{k_1,k_2,\dots,k_d}$ in which all factor graphs are rings.)

3.5.2 Congestion and Central Cut of Rings and XTs

The congestion and central cut of both a ring and an XT are needed to calculate the bounds obtained in Section 3.3. We present the following lemma for the congestion of a ring.

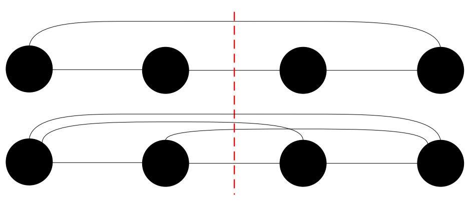
Lemma 6. *The congestion of* R_k *with multiplicity* $r = 2$ *has two possible upper bounds depending on whether the number of vertices* k *is even or odd, as follows,*

$$m_2(R_k) \leq \begin{cases} \frac{k^2}{4} & \text{if } k \text{ is even} \\ \frac{k^2-1}{4} & \text{if } k \text{ is odd} \end{cases} \tag{3.9}$$

Proof: While a path had only one possible routing, for R_k we have two possible routes connecting each pair of nodes. When we embed rK_k, for $r = 2$, into R_k, we embed the parallel edges connecting two nodes through the shortest path, when this is unique. Otherwise, when two nodes are equally distant along each of the two available routes (note that this happens only if k is even), each parallel edge is embedded following a different route.

A counting argument yields the congestion on any edge under this routing. Let us consider without loss of generality the edge $e = (0, 1)$. Any two nodes that are at distance at most $\lfloor \frac{k-1}{2} \rfloor$ by a shortest path that crosses e, have the two parallel edges connecting them embedded in this path. Then, there are $\lfloor \frac{k-1}{2} \rfloor$ shortest paths that cross e and end at node 1, $\lfloor \frac{k-1}{2} \rfloor - 1$ shortest paths that cross e and end at node 2, and so on. Hence, the congestion in e due to the embedding of these edges is

$$2 \sum_{i=1}^{\lfloor \frac{k-1}{2} \rfloor} \left\lfloor \frac{k-1}{2} \right\rfloor - i + 1 = \left\lfloor \frac{k-1}{2} \right\rfloor \left(\left\lfloor \frac{k-1}{2} \right\rfloor + 1 \right).$$


(a) The 4-vertex ring and clique.

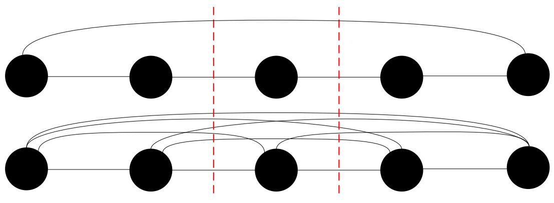

(b) The 5-vertex ring and clique.

Figure 3.3: Rings and possible cuts.

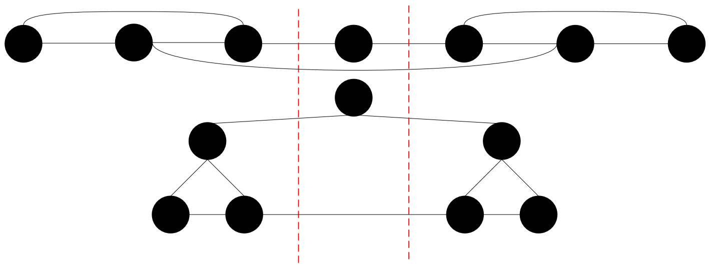


Figure 3.4: Central cut on a extended complete binary tree.

When k is odd, this is the congestion of the edge e, which becomes

$$\left(\frac{k-1}{2}\right)\left(\left(\frac{k-1}{2}\right)+1\right)=\frac{k-1}{2}\frac{k+1}{2}=\frac{k^2-1}{4}.$$

When k is even, edge e is also crossed by one of the parallel embedded edges connecting nodes at distance $k/2$. This increases the congestion in edge e by $k/2$. Hence, given that $\lfloor(k-1)/2\rfloor = (k-2)/2$, the congestion for k even is

$$\left(\frac{k-2}{2}\right)\left(\left(\frac{k-2}{2}\right)+1\right)+\frac{k}{2}=\frac{k^2}{4}.$$

■

Corollary 6. *The normalized congestion with multiplicity $r = 2$ of a ring is $\beta_2(R_k) = 1/4$.*

Similarly to what happened with paths and CBTs, the congestion of rings and XTs is the same. The extended complete binary tree X_k has a Hamiltonian cycle [65], so we can find a ring R_k contained onto it. Consequently, the congestion of an XT and a ring with the same number of nodes will be the same.

Corollary 7. *The normalized congestion with multiplicity $r = 2$ of an XT is $\beta_2(X_k) = 1/4$.*

Due to these similarities, central cuts of both graphs are also going to be the same. As can be easily deduced from Figures 3.3 and 3.4, $CC(R_k) = CC(X_k) = 2$.

3.5.3 Bounds on the BW of Products of XTs and Rings

With the normalized congestion and central cut of the different factor graphs, we can calculate the lower and upper bounds on the bisection width of products of XTs and rings. We will start by the lower bound on the bisection width presenting the following lemma.

Lemma 7. *The bisection width of a d-dimensional mesh-connected XTs and rings, $MCXR^{(d)}$, is lower bounded by* $2\Psi(\alpha)$.

Proof: The normalized congestion of both factor graphs is $\beta_2(R_k) = \beta_2(X_k) = 1/4$. Then, applying Corollary 2 with $r = 2$, we get $2\Psi(\alpha)/(4(1/4)) \leq BW(MCXR^{(d)})$. This yields, $BW(MCXR^{(d)}) \geq 2\Psi(\alpha)$. ■

We calculate now the upper bound on the bisection width of a d-dimensional mesh-connected rings and XTs.

Lemma 8. *The bisection width of a d-dimensional, $MCXR^{(d)}$, is upper bounded by* $2\Psi(\alpha)$.

Proof: The d-dimensional mesh-connected XTs and rings graph can also be embedded into a d-dimensional array, so then, we can use Theorem 4. As happened with the congestion, the value of the central cut of both XTs and rings is the same, concretely, $CC(R_k) = CC(X_k) = 2$, independently of their sizes or number of levels. Hence, $\max_i\{CC(G_{k_i})\} = 2$ (where G_{k_i} is either a ring or an XT). Then, $BW(MCXR^{(d)}) \leq 2\Psi(\alpha)$. ■

From Lemma 7 and Lemma 8, Theorem 6 follows.

Theorem 6. *The bisection width of a d-dimensional mesh-connected extended trees and rings $MCXR^{(d)}_{k_1,k_2,\ldots,k_d}$ is* $2\Psi(\alpha)$.

From the bisection width of the d-dimensional mesh-connected XTs and rings, we can derive the following corollaries for the particular cases where all the factor graphs are rings, Torus $T^{(d)}$, or XTs, mesh-connected extended trees $MCX^{(d)}$.

Corollary 8. *The bisection width of the d-dimensional torus $T^{(d)}_{k_1,k_2,\ldots,k_d}$ is $BW(T^{(d)}) = 2\Psi(\alpha)$.*

Corollary 9. *The bisection width of the d-dimensional mesh-connected extended trees $MCX^{(d)}_{k_1,k_2,\ldots,k_d}$ is $BW(MCX^{(d)}) = 2\Psi(d)$.*

3.6 BCube

We devote this section to obtain bounds on the bisection width of a d-dimensional BCube [79]. BCube is different from the topologies considered in the previous sections because it is obtained as the combination of basic networks formed by a collection of k nodes (servers) connected by a switch. These factor networks are combined into multidimensional networks in the same way product graphs are obtained from their factor graphs. This allows us to study the BCube as an special instance of a product network. The d-dimensional BCube can be obtained as the d dimensional product of one-dimensional BCube networks, each one of k nodes.

3.6.1 Factor and Product Graphs

We first define a *Switched Star network* and how a d-dimensional BCube network is built from it.

Definition 16. *A* Switched Star network *of k nodes, denoted SS_k, is composed of k nodes connected to a k-ports switch. It can be seen as a complete graph K_k where all the edges have been replaced by a switch.*

An example of SS_5 is presented in Figure 3.5(a). Combining d copies of SS_k as factor networks of the Cartesian product, we obtain a d-dimensional BCube.

Definition 17. *A* d-dimensional BCube*, denoted by $BC_k^{(d)}$, is the Cartesian product of d SS_k (the switches are not considered nodes for the Cartesian product). I.e., $BC_k^{(d)} = SS_k \times SS_k \times \cdots \times SS_k$.*

The topology of $BC_5^{(2)}$ (which results from combining 2 copies of SS_5) is shown in Figure 3.6. $BC_k^{(d)}$ can also be seen as a d-dimensional homogeneous array where all the edges in each path have been removed and replaced by a switch where two nodes $(u_1, ..., u_i, ..., u_d)$ and $(v_1, ..., v_i, ..., v_d)$ are connected to the same switch if and only if $(u_i \neq v_i)$ and $u_j = v_j$ for all $j \neq i$.

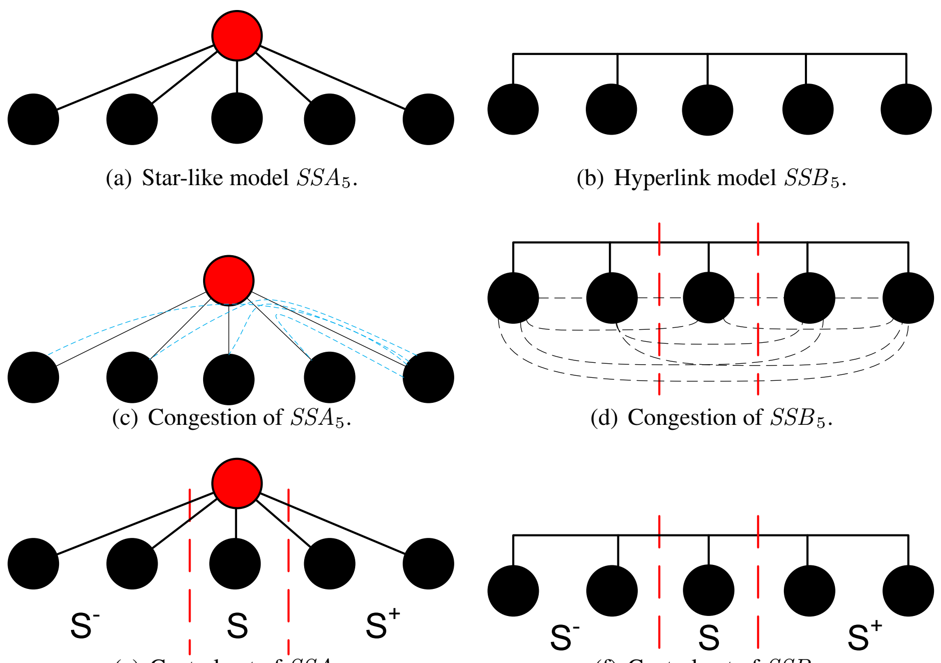
Strictly speaking, the bisection width of $BC_k^{(d)}$ is the smallest number of links (connecting nodes to switches) that have to be removed to bisect it (extending the definition to networks with switches). However, the main reason for obtaining the bisection width of a d-dimensional BCube is to be able to bound its bisection bandwidth. However, as the d-dimensional BCube is not a typical graph, the bisection width can have different forms depending on where the communication bottleneck is located in a BCube network.

We present two possible models for SS_k. The first one, Model A or *star-like model,* denoted by SSA_k, consists of k nodes connected one-to-one to a virtual node which represents the switch. This model corresponds with the actual physical topology of BCube. The second one, Model B or *hyperlink model,* denoted by SSB_k, consists of k nodes connected by a hyperlink[4]. While the two presented models are logically equivalent to a complete graph, they have a different behavior from the traffic point of view. We show this with two simple examples.

Let us consider that we have a SS_3 where the links have a speed of 100 Mbps while the switch can switch at 1 Gbps. Under these conditions, the links become the bottleneck of the network and, even when the switches would be able to provide a bisection bandwidth of 1 Gbps, the effective bisection bandwidth is only of 200 Mbps in both directions.

Consider another situation now, where the BCube switch still supports 1 Gbps of internal traffic but the links also transmit at 1 Gbps. In this case, the switches are the bottleneck of the network and the bisection bandwidth is only 1 Gbps, although the links would be able to support up to 2 Gbps.

[4]This model is quite similar to the one proposed by Pan in [136].



(a) Star-like model SSA_5.
(b) Hyperlink model SSB_5.
(c) Congestion of SSA_5.
(d) Congestion of SSB_5.
(e) Central cut of SSA_5.
(f) Central cut of SSB_5.

Figure 3.5: Proposed models for a 5-node switched star, SS_5, their congestion and central cut.

The first example illustrates an scenario where we would bisect the network by removing the links that connect the servers to the switches, which corresponds to Model A. On the other hand, what we find in the second example is a typical scenario for Model B, where we would do better by removing entire switches when bisecting the network. In particular, being s the switching capacity of a switch, and T the traffic supported by a link, we will choose Model A when $s \geq \lfloor \frac{k}{2} \rfloor \cdot 2T$ and Model B when $s \leq 2T$. (Note that this does not cover the whole spectrum of possible values of s, T, and k.)

3.6.2 Congestion and Central Cut of BCube

We will compute now the congestion and central cut of both models in order to be able to calculate the respective lower and upper bounds. We start by the congestion and central cut of Model A. If we set $r = 1$, the congestion of every link of the star is easily found[5] to be $m_r(SSA_k) = k - 1$ as shown in Figure 3.5(c). The central cut, which is also trivial, can be found in Figure 3.5(e). Both will depend on whether the number of nodes k is even or odd.

Corollary 10. *The normalized congestion of SSA_k is $\beta_r(SSA_k) = \frac{k-1}{k^2-b}$, and the central cut is $CC(SSA_k) = \frac{k-b}{2}$, where $b = k \bmod 2$.*

Having computed the congestion and the central cut for Model A, we will compute them now for Model B. If we set $r = 1$ there will be only one edge to be removed, the congestion of the

[5]Note that in the computation of the congestion, the switch is not considered a node of the graph.

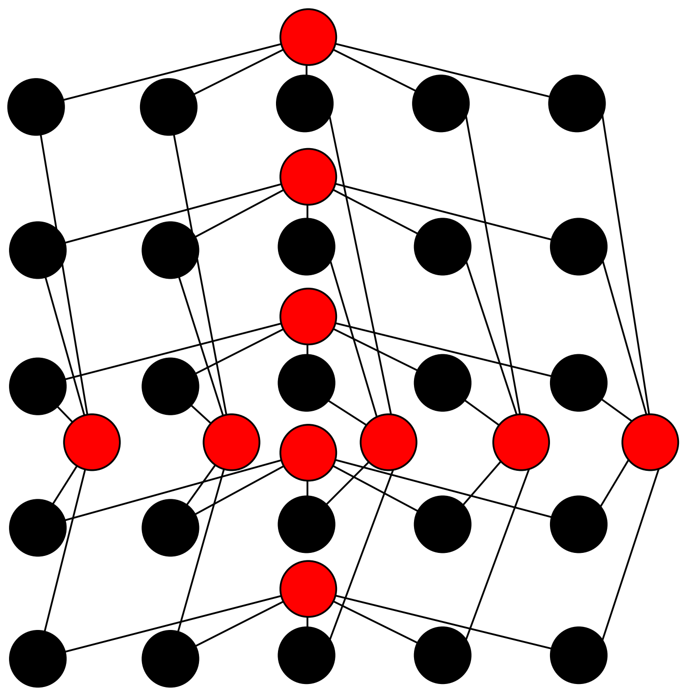


Figure 3.6: Cartesian product of two SSA_5 networks.

graph will be total amount of edges of its equivalent K_k, i. e., $m_r(SSB_k) = \frac{k(k-1)}{2}$. The central cut is also easily computed, as there is only one hyperlink. Both $m_r(SSB_k)$ and $CC(SSB_k)$ are shown in Figure 3.5.

Corollary 11. *The normalized congestion of* SSB_k *is* $\beta_r(SSB_k) = \frac{k-1}{2(k^2-b)}$*, where* $b = k \bmod 2$*, and the central cut is* $CC(SSB_k) = 1$.

3.6.3 Bounds on the BBW of BCube

Having computed the congestion and central cut of both models, we can calculate the lower and upper bounds on the bisection width of each one of them. We will start by the lower and upper bounds on the bisection width of Model A and, then, we will calculate both bounds for Model B. We first present the following lemma for the lower bound on the bisection width of a Model A BCube.

Lemma 9. *The bisection width of a Model A* d*-dimensional BCube,* $BCA_k^{(d)}$*, is lower bounded by* $\frac{k^{d+1}}{4(k-1)}$ *if* k *is even, and by* $\frac{k+1}{4}\frac{k^d-1}{k-1}$ *if* k *is odd.*

Proof: Using the value of the normalized congestion of a Model A BCube in Corollary 2, it follows that

$$BW(BCA_k^{(d)}) \geq \begin{cases} \frac{1}{4}\frac{k^2}{k-1}\Psi(\alpha) = \frac{k^{d+1}}{4(k-1)} & \text{if } k \text{ is even} \\ \frac{k+1}{4}\Psi(\alpha) = \frac{k+1}{4}\frac{k^d-1}{k-1} & \text{if } k \text{ is odd} \end{cases}$$

■

After presenting the lower bound on the bisection width of a Model A d-dimensional BCube, we follow with the upper bound.

Lemma 10. *The bisection width of a Model A d-dimensional BCube, $BCA_k^{(d)}$, is upper bounded by $\frac{k^d}{2}$ if k is even, and by $\frac{k^d-1}{2}$ if k is odd.*

Proof: The Cartesian product of Model A star-like factor graphs can be embedded into a d-dimensional array, so Theorem 4 will be extremely useful again. If we use the values of the central cut of Model A in Theorem 4, is immediate to compute the following upper bound

$$BW(BCA_k^{(d)}) \leq \begin{cases} \frac{k^d}{2} & \text{if } k \text{ is even} \\ \frac{k^d-1}{2} & \text{if } k \text{ is odd.} \end{cases}$$

■

Now, from the combination of Lemma 9 and Lemma 10 we can state Theorem 7:

Theorem 7. *The value of the bisection bandwidth of a Model A d-dimensional BCube, $BCA_k^{(d)}$, is in the interval $[2T \cdot \frac{k^{d+1}}{4(k-1)}, 2T \cdot \frac{k^d}{2}]$ if k is even, and in the interval $[2T \cdot \frac{k+1}{4}\frac{k^d-1}{k-1}, 2T \cdot \frac{k^d-1}{2}]$ if k is odd.*

From the observation that the topology of BCube is the same as that of Model A, we derive the following corollary.

Corollary 12. *The bisection width of a d-dimensional BCube $BC_k^{(d)}$ satisfies,*

$$BBW(BCA_k^{(d)}) \in \begin{cases} [\frac{k^{d+1}}{4(k-1)}, \frac{k^d}{2}] & \textit{if } k \textit{ is even} \\ [\frac{k+1}{4}\frac{k^d-1}{k-1}, \frac{k^d-1}{2}] & \textit{if } k \textit{ is odd.} \end{cases}$$

Let us calculate now the bounds of a Model B d-dimensional BCube. As we did with Model A, we will first prove the lower bound and then the upper one. For the lower bound we present the following lemma.

Lemma 11. *The bisection width of a Model B d-dimensional BCube, $BCB_k^{(d)}$, is lower bounded by $\frac{k^d}{2(k-1)}$ if k is even, and by $\frac{k+1}{2k}\frac{k^d-1}{k-1}$ if k is odd.*

Proof: Like in the case of Model A, we use the value of the normalized congestion of Model B in Corollary 2. Since all the dimensions have the same size k, it follows that

$$BW(BCB_k^{(d)}) \geq \begin{cases} \frac{1}{4}\frac{2k}{k-1}\Psi(\alpha) = \frac{k^d}{2(k-1)} & \text{if } k \text{ is even} \\ \frac{1}{4}\frac{2(k+1)}{k}\Psi(\alpha) = \frac{k+1}{2k}\frac{k^d-1}{k-1} & \text{if } k \text{ is odd} \end{cases}$$

■

We present now Lemma 12 for the upper bound on the bisection width of a Model B d-dimensional BCube.

Table 3.3: Gap between bounds (results in Gbps)

		Model A			Model B		
k	d	BBW	LB	UB	BBW	LB	UB
3	2	0.8	0.8	0.8	4	2.667	4
4	2	1.6	1.067	1.6	5	3.125	5
7	2	4.8	3.2	4.8	8	4.571	8
8	2	6.4	3.657	6.4	8	4.571	9
3	3	2.6	2.6	2.6	13	8.667	13
4	3	6.4	4.267	6.4	16	10.667	21
7	3	34.2	22.8	34.2	57	32.571	57
8	3	51.2	29.257	51.2	64	36.571	73

Lemma 12. *The bisection width of a Model B d-dimensional BCube, $BCB_k^{(d)}$, is upper bounded by $\frac{k^d-1}{k-1}$.*

Proof: As for Model A, the d-dimensional BCube resulting from the Cartesian product of Model B graphs can be embedded into a d-dimensional array. Thanks to this fact, we can use the computed value of its central cut in Theorem 4 to obtain the upper bound on the bisection width, $BW(BCB_k^{(d)}) \leq 1 \cdot \Psi(\alpha) = \frac{k^d-1}{k-1}$. ■

Combining the previous lemmas we can state the following theorem.

Theorem 8. *The value of the bisection bandwidth of a Model B d-dimensional BCube, $BCB_k^{(d)}$, is in the interval $[s \cdot \frac{k^d}{2(k-1)}, s \cdot \frac{1-k^d}{1-k}]$ if k is even, and in the interval $[s \cdot \frac{k+1}{2k}\frac{k^d-1}{k-1}, s \cdot \frac{k^d-1}{k-1}]$ if k is odd.*

In Table 3.3 we present some results for some concrete BCube networks. We assume the same 2 scenarios we used in the previous example: links of $T = 100$ Mbps and switches with $s = 1$ Gbps for Model A ($s \geq \lfloor \frac{k}{2} \rfloor \cdot 2T$); and links of $T = 1$ Gbps and switches with $s = 1$ Gbps for Model B ($s \leq 2T$).

3.7 Conclusions

Exact results for the bisection bandwidth of various d-dimensional classical parallel topologies have been provided in this chapter. These results consider any number of dimensions and any size, odd or even, for the factor graphs. These multidimensional graphs are based on factor graphs such as paths, rings, complete binary trees or extended complete binary trees. Upper and lower bounds on the bisection width of a d-dimensional BCube are also provided. Some of the product networks studied had factor graphs of the same class, like the d-dimensional torus, mesh-connected trees or mesh-connected extended trees, while some other combined different factor graphs, like the mesh connected trees and paths or mesh-connected extended trees and rings. See Table 3.1 for a summary of the results obtained.

Chapter 4

Reducing Wiring in Data Centers. A Simulated Annealing Approach

4.1 Overview

In this chapter we study a version of the Multidimensional Arrangement Problem (MAP) that embeds a graph into a multidimensional array minimizing the aggregated (Manhattan) distance of the embedded edges. This problem includes the minimum Linear Arrangement Problem (minLA) as a special case, among others.

We propose JAM, a tabu-based two-stage simulated annealing heuristic for this problem. JAM uses a novel median-based neighborhood function and non-trivially adapts multiple techniques from the minLA literature to work for multiple dimensions. We describe each one of the JAM components in detail.

Due to the scarcity of specific benchmarks for MAP, the performance of JAM has been tested with benchmarks for the minLA and the Quadratic Assignment Problem (QAP). In particular, more than 80 different instances, either weighted or unweighted, have been used. It is worth to notice that the host graph in minLA is a one-dimensional array while for QAP is, in general, a 2 dimensional graph. JAM obtains the optimal or best known result in most of the problem instances and even improves the results of some minLA instances.

Finally, the results in this chapter are not limited to the original minLA and QAP instances. From these instances we generate a whole benchmark for MAP, defining instances for 1, 2 [1] and 3 dimensions, enlarging, hence, the benchmark resources for the research community. The results obtained by JAM for these set of instances are also provided as a reference.

RoadMap The chapter is organized as follows. In Section 4.2 we present our own algorithm as well as a detailed description of its different elements. In Section 4.3 we present the numerical

[1]We prepared two different sets of instances for the 2-dimensional case. The first one is restricted to the case in which guest and host graphs have the same size, i.e., where $|V'| = |V|$. The second one is for a deployment which is more compact (square) and that allows having extra locations, i.e., where $|V'| \geq |V|$.

Algorithm 1: JAM Pseudo Code

```
$\varphi^* \leftarrow$ SetInitialSolution();
$\varphi_{best} \leftarrow \varphi^*$;
$k \leftarrow 0$; $T(k) \leftarrow$ SetInitialTemperature();
while the termination criteria do not hold do
    for the predefined number of iterations at temperature $T(k)$ do
        Choose a node $u$ uniformly at random;
        with probability $p_N$: set $L \leftarrow N^u$;
        else set $L \leftarrow \{l\}$, where $l$ is a location chosen uniformly at random;
        Discard all locations $l \in L$ that would lead to a move in the tabu list;
        $\Phi \leftarrow \{\varphi_l : \varphi_l$ is the arrangement after moving $u$ to $l$ in $\varphi^*$, $\forall l \in L\}$;
        $\varphi' \leftarrow \arg\min_{\varphi \in \Phi}\{C(\varphi)\}$;
        if $C(\varphi') > C(\varphi^*)$ then
            $\varphi' \leftarrow \varphi$ chosen from $\Phi$ with probability proportional to $C(\varphi)$;
        $\delta \leftarrow C(\varphi') - C(\varphi^*)$;
        with probability $e^{-\delta/T(k)}$: $\varphi^* \leftarrow \varphi'$; $\varphi_{best} \leftarrow \arg\min\{C(\varphi_{best}), C(\varphi^*)\}$;
    $k \leftarrow k + 1$; $T(k) \leftarrow$ UpdateTemperature($T(k-1)$);
    GuidedRestart($\varphi^*$, $\varphi_{best}$);
```

results obtained with our heuristic for multiple benchmarks from the minLA and QAP literature and for which we provide results in 1, 2 and 3 dimensions. We finish the chapter by presenting some conclusions in Section 4.4.

4.2 The Algorithm JAM

In this section we present JAM, a tabu based two-stage simulated annealing algorithm applied to MAP. First, we introduce the notation used throughout the rest of the chapter, then provide an overview of JAM, and finally describe each one of its elements.

4.2.1 Notation

Given a graph $G = (V, E)$, V and E denote its sets of vertices and edges, respectively. Each edge $(i, j) \in E$ has an associated weight denoted by w_{ij}. We denote by A^u the set of nodes adjacent to node u in G.

The host graph $H(V', E')$ is a D-dimensional array. Recall that $|V'| \geq |V|$. The nodes of H are called *locations*. Each location $l \in V'$ is defined by a D-vector $(l_1, l_2, \ldots, l_D)$, where l_i is the dimension i coordinate of location l. On the other hand, $d_i(X)$ is used to denote the dimension i coordinate of all elements of a set X of locations.

In order to improve the cost $C(\varphi)$ of an arrangement φ, JAM performs node "movements." By movement we refer to the action of "moving" node u from a location l to a location l', and "moving" the node v, if any, which is at location l' to l. Formally, this means transforming the arrangement φ into a new φ' such that $\varphi'(x) = \varphi(x)$ for all $x \in V \setminus \{u, v\}$, $\varphi'(u) = l'$, and

$\varphi'(v) = l$. There is only a set of valid locations to which a node u can move (see Section 4.2.3 below), this set is called its *neighborhood* and is denoted N^u.

4.2.2 Overview of JAM

A sketch of JAM is provided as pseudo code in Algorithm 1. Similarly, a flow diagram can be found in Figure 4.1. JAM is a two-stage heuristic whose first stage provides an initial solution based on the McAllister heuristic [113]. This arrangement is also the initial best known solution. The current solution and the best known solution are stored in φ^* and φ_{best}, respectively (Lines $1-2$). Then, the initial temperature $T(0)$ (Line 3) for the SA is computed. After that, the cooling down process starts, which will take place until the termination criteria are met (Line 4). For every temperature $T(k)$, JAM runs a predefined number of iterations (Line 5), starting with $T(0)$. In each iteration, a node u to be moved is chosen uniformly at random (Line 6). Then, with

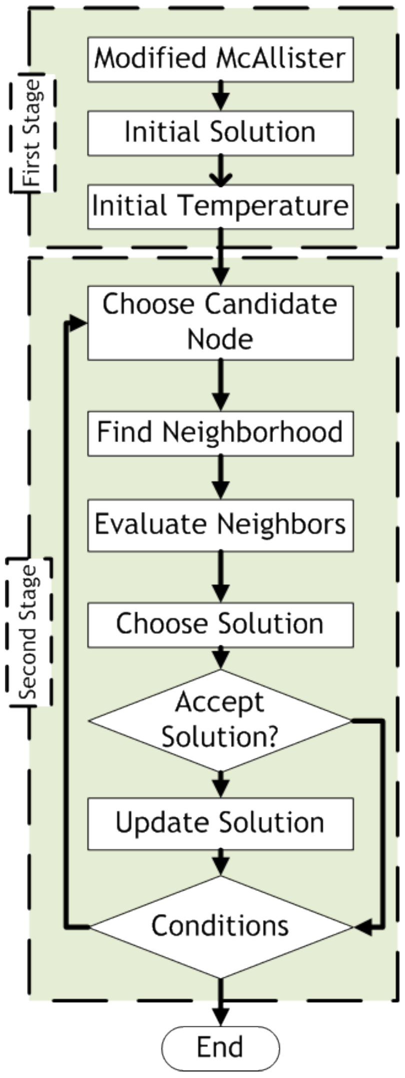


Figure 4.1: Flow diagram of JAM.

probability p_N, the set of locations L to which u may be moved is chosen to be the neighborhood N^u (Line 7). Otherwise the only location that will be considered is a randomly chosen one l (Line 8). Not explicitly shown in Algorithm 1, JAM maintains a tabu list of movements that are not to be redone. Hence, all elements in L that lead to a move in this tabu list are discarded (Line 9). Now, the arrangements Φ resulting of moving u to the remaining locations in L are obtained (Line 10), and the arrangement φ' with the lowest cost among them is chosen (Line 11). If the cost of this φ' is larger than the cost of the current solution φ^*, φ' is replaced by an arrangement chosen from Φ with probabilities proportional to their respective costs (Line $12-13$). To complete the iteration, the proposed φ' is adopted with probability $e^{-\delta/T(k)}$, which implies updating φ^* and, if corresponds, φ_{best} (Line 15). (Observe that if $\delta < 0$ then φ' is always adopted.) Once the given number of iterations for $T(k)$ is reached, it is updated (Line 16) and it is decided whether resetting φ^* to φ_{best} is needed (Line 17).

4.2.3 Elements of JAM

We now provide a detailed description of the elements mentioned above that conform JAM.

4.2.3.1 First Stage: Initial Solution

McAllister heuristic [113] has been adapted to multiple dimensions and used to obtain an initial solution. McAllister's is a greedy heuristic based on a frontal increase minimization strategy. It chooses a starting node at random and maps it to some location. Then, it greedily maps the rest of nodes. To do so, it maintains three sets of nodes U (Unplaced), P (Placed) and F (Front, the set of placed nodes with at least one neighbor in set U). The next node to be mapped is the one with the least neighbors in set $U \setminus F$, so the front set is minimized.

We implement McAllister's heuristic with all the proposed refinements: a tie breaking strategy, improved initial node selection and deferred node placement (We refer the reader to [113] for further details). Then, we devised a multidimensional allocation technique which maps the first node to location $(0, 0, \ldots, 0)$ in the host graph, and then greedily decides which of the neighboring locations to those of the already allocated nodes is the best position, in terms of cost, for the next node. One example of how this allocation technique works in 2 dimensions is provided in Figure 4.2.

4.2.3.2 Initial Temperature

We decided to initialize the temperature using the same method as Tello *et al.* [139], which employs the technique proposed by Varanelli and Cohoon [154]. This method approximates the simulated annealing temperature $T(k)$ at which a solution φ^* with cost $C(\varphi^*)$ can be found as

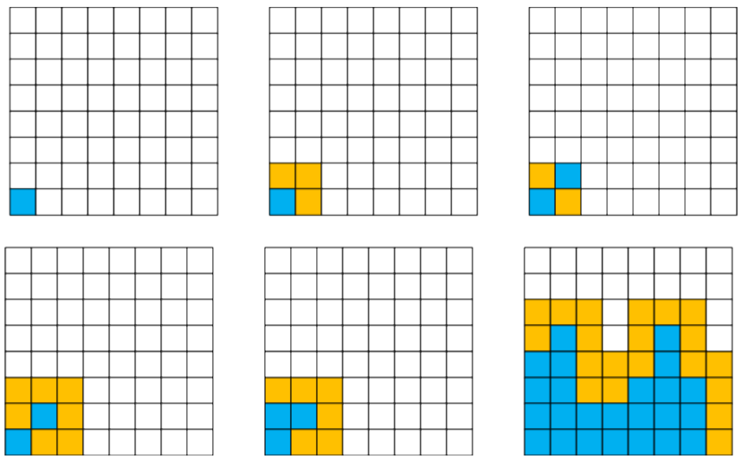


Figure 4.2: Example of a possible evolution for the multidimensional McAllister allocation heuristic in 2 dimensions. The allocated nodes are represented in blue, the nodes belonging to the front in yellow.

best solution. Hence the initial temperature is given by[2]

$$T(0) \approx \left| \frac{\sigma_\infty^2}{C_\infty - C(\varphi^*) - \gamma_\infty \sigma_\infty} \right|,$$

where C_∞ and σ_∞ represent the expected cost and average deviation of the cost over the solution space; $C(\varphi^*)$ represents the cost of the initial solution and γ_∞ represents the difference between the expected cost C_∞ and the best known solution φ_{best} at temperature $T(k)$. γ_∞ can be calculated probabilistically from the number of iterations predefined at each temperature. We refer the reader to [154] for further details.

4.2.3.3 Cooling Schedule

Our cooling schedule is based on the work from Aarts and Korst [7]. They propose a statistical cooling schedule which depends on the previous temperature, the average deviation of the solutions obtained with the previous temperature $\sigma_{T(k-1)}$, and a tuning parameter λ (such that for small values of λ we obtain small temperature reductions). The cooling schedule is given by the following equation:

$$T(k) = T(k-1)\left(1 + \frac{\log(1+\lambda)T(k-1)}{3\sigma_{T(k-1)}}\right)^{-1}.$$

4.2.3.4 Neighboring Solutions

In order to reduce the search space of locations to which a certain node u can be moved, we define a median-based neighborhood function. This function returns a set of neighbors N^u, which

[2]We use the absolute value of the expression, unlike in [154], to deal with some cases that resulted in negative values.

is the set of contiguous locations that will be considered for the movement of u. Intuitively, we choose the set N^u to be the locations that minimize the cost of the edges incident in u assuming that only u changes its location (in this fictitious arrangement u may share location with other nodes).

We describe now the process we use to obtain N^u. Let us assume that the nodes adjacent to u in G are $A^u = \{v^1, v^2, \dots, v^n\}$, and that their respective current location is $l^j = \varphi^*(v^j), \forall j \in [1, n]$. Let us fix one dimension $i \in [1, D]$, and let us sort the nodes in A^u by the dimension i coordinate of their current location, so that $l_i^{j_1} \leq l_i^{j_2} \leq \cdots \leq l_i^{j_n}$. Then, we compute the smallest $m \in [1, n]$ that satisfies

$$\Delta(m) = \sum_{k=1}^{m} w_{uv^{j_k}} - \sum_{k=m+1}^{n} w_{uv^{j_k}} \geq 0.$$

If $\Delta(m) = 0$ then we define a range of values $r_i = [l^{j_m}, l^{j_{m+1}})$. Otherwise, if $\Delta(m) > 0$ then we define the range as the singleton value $r_i = [l^{j_m}]$.

After applying this method to each dimension separately we have ranges $r_1, r_2, \dots, r_D$. The D-polytope obtained by the combination of these ranges is the set of locations in N^u. I.e., all locations l such that $l_i \in r_i, \forall i \in [1, D]$ belong to N^u. In our implementation of JAM we extended N^u with all the locations that are within distance 2 of the set described, just to increase the movement options.

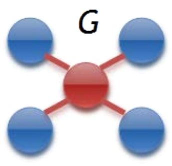
Figure 4.3 shows how our neighboring function works for a 2-dimensional graph. In it we can see, in Subfigure 4.3(a), a subset of nodes G formed by the candidate node, depicted in red, and its set of neighbors, in blue. Then, in Subfigure 4.3(b), we have where each one of these nodes are located in the host graph H and the ranges r_1 and r_2 that define the sets of ideal locations for each dimension. Finally, in Subfigure 4.3(c), we can see the 2-dimensional polytope that defines the set of ideal locations which become candidates to host the candidate node.

4.2.3.5 Evaluating Solutions

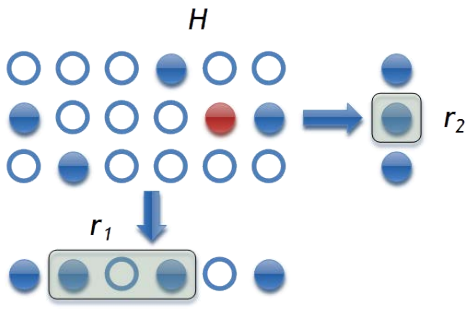
We defined the cost of an arrangement $C(\varphi)$ in Eq. 2.1. However, two different solutions might have the same cost. To consider these cases, we use instead a cost function $C'(\varphi)$ introduced in [140]. The authors there proposed a refined method for estimating the cost of solutions in a minLA problem which considers not only the cost derived from the paths in H but also how the costs of these paths are distributed. The cost of an arrangement φ is then given by

$$C'(\varphi) \quad = \quad \sum_{k=1}^{\Theta} \left(k + \frac{n!}{(n+k)!} \right) e_k, \tag{4.1}$$

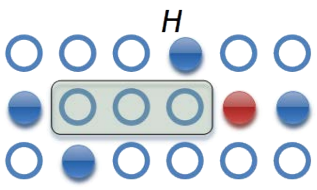
where $\Theta = \sum_{i=1}^{D} d_i - 1$ is the diameter of H and e_k is the number of paths of length k in H. Note that the second term of this formula is always smaller than 1. Then, for solutions where the



(a) Subset of nodes G.



(b) Locations in Host graph H.



(c) 2-dimensional polytope of ideal locations.

Figure 4.3: Example of JAM neighboring function in 2 dimensions.

cost would be the same if we had only considered the first term, the total cost will be smaller if the arrangement has longer paths. A solution with a larger number of longer paths is preferred as it would be, in principle, easier to improve.

4.2.3.6 Tabu Search (TS)

In order to favor the exploration abilities of JAM we incorporate TS. As we said, moving a node u from position l to position l' implies moving the node v in position l', if any, to position l. Our TS mechanism will check that neither u nor v have been in locations l or l', respectively, during the last T_s moves, being T_s the size of our tabu list.

To control when a move is tabu we use a $|V| \times |V'|$ matrix. Every time a node is moved to a certain location, we store the iteration number at which that move was done. If a proposed move has been done during the last T_s iterations, it is discarded. There is one exception to this rule, the *aspiration criterion*. We implemented the most common one: a move will be accepted, despite of being tabu, when it leads to a smaller $C'(\varphi_{best})$.

4.2.3.7 Guided Restarts

We implement guided restarts in order to help the algorithm to escape from some strong local minima. A restart consists in resetting φ^* to φ_{best}. We decide if a restart is needed after finishing all the iterations at a certain temperature. A restart occurs with probability

$$P(restart) = 1 - e^{(-\frac{|\varphi^* - \varphi_{best}|}{\varphi_{best}} \frac{T(0)}{T(k)} \gamma)},$$

where $T(0)$ and $T(k)$ are the initial and current temperatures, and γ is a tuning parameter that depends on the size of the graph.

4.2.3.8 Termination Criteria

We use two termination criteria. Our algorithm will stop when (1) $T(k)$ goes below a predefined temperature threshold T_{th} or when (2) the percentage of accepted moves improving φ^* while at temperature $T(k)$ goes below a second predefined threshold P_{th}. The values for these thresholds depend on the size of the graph.

4.3 Evaluation of JAM

In this section we present the results obtained for a set of benchmark instances ran in order to evaluate JAM's performance. Ideally, we would have used a set of instances for which we had results in multiple dimensions. However, due to non-existence, to the best of our knowledge, of such a set of instances, we used graphs belonging to benchmarks from the minLA and QAP literature. Our intention, however, is two-fold. First, we want to create such a collection of instances so they can be used in future MAP works as benchmark. Second, by running these graphs in 1 and 2 dimensions, we are broadening the available number of instances and results for both minLA and QAP benchmark collections.

JAM results for graphs from both collections of instances are presented in Tables 4.1, 4.2, 4.3 and 4.4. We provide two different results for 2 dimensions. First, with either $|V'| = |V|$ or the original configuration (for the case of QAPLIB instances). Second, for a more compact layout allowing that $|V'| \geq |V|$. The *BKV* (Best Known Value) and a δ (the difference between JAM's result and the BKV in percentage) are provided when a BKV is available. In particular, they are given for 1 dimension results in Table 4.1 and for the first results in 2 dimensions in Table 4.4. For the 3-dimensional host graphs we chose the number of nodes in each dimension so that the number of empty locations is minimized. In these tables R, C and D denote the number of nodes in the respective dimensions of H (the letters come from rows, columns and depth).

We provide results for 81 different graphs. We ran JAM a minimum of 5 times per instance, for the sake of statistical significance. We used different configurations that depended on the number of edges of the graph. In particular, the parameters being changed were the predefined number of iterations per temperature, T_{th}, P_{th} and γ. The values used can be found in Table 4.5. Other parameters used during the experiments which fixed for all the graph instances were the probability p_N, fixed at a 0.9; λ, which was fixed to 0.1; and T_s, which was fixed to $2 \cdot |V|$.

Table 4.1: Results for the minLA benchmark instances with 1 & 3 dimensions

Graph	$\lvert V \rvert$	$\lvert E \rvert$	**1D**			**3D**			
			BKV	δ	Cost	R	C	D	Cost
bcspwr01	39	46	**106**	0%	106	2	4	5	50
bcspwr02	49	59	**161**	0%	161	2	5	5	66
bcspwr03	118	179	[588, 679]	**-2,50%**	662	4	5	6	225
bcsstk01	48	176	**1132**	1,77%	1152	2	4	6	314
bcsstk02	66	2145	47916	**-0,02%**	47905	2	3	11	10945
bcsstk04	132	1758	[27569, 29804]	0,03%	29812	3	4	11	5192
bcsstk05	153	1135	[9653, 11057]	0,02%	11059	2	7	11	2895
bcsstk22	110	254	-	-	981	2	5	11	353
bintree10	1023	1022	3696	0%	3696	3	11	31	1098
c1y	828	1749	62230	0.33%	62436	6	6	23	3962
can___144	144	576	[2304, 3224]	0%	3224	4	6	6	990
can___161	161	608	[5657, 6696]	0%	6696	4	6	7	1012
can___24	24	68	**210**	0%	210	2	3	4	98
can___61	61	248	**1137**	0%	1137	3	3	7	425
can___62	62	78	[187, 212]	**-0,94%**	210	3	3	7	84
can___73	73	152	[971, 1100]	0%	1100	3	5	5	229
can___96	96	336	[2105, 2702]	0%	2702	4	4	6	525
curtis54	54	124	**454**	0%	454	3	3	6	179
dwt___162	162	510	[2032, 2431]	0,25%	2437	3	6	9	766
dwt___209	209	767	[5905, 6387]	20,78%	7714	5	6	7	1258
dwt___221	221	704	[3603, 3779]	**-0,13%**	3774	5	5	9	1062
dwt___245	245	608	[3422, 3860]	4,53%	4035	5	7	7	920
dwt___59	59	104	**289**	0%	289	3	4	5	128
dwt___66	66	127	**192**	0%	192	2	3	11	159
dwt___72	72	75	**167**	0%	167	3	4	6	80
dwt___87	87	227	**932**	0%	932	2	4	11	334
fidap005	27	126	**414**	0%	414	3	3	3	242
fidapm05			**1003**	0%	1003	2	3	7	487
gd95c	62	144	**506**	0%	506	3	3	7	210
gd96b	111	193	1416	0%	1416	4	4	7	380
gd96c	65	125	**519**	0%	519	2	3	11	166
gd96d	180	228	2391	0%	2391	5	6	6	382
ibm32	32	90	**485**	0%	485	2	4	4	155
impcol_b	59	281	[1810, 2076]	0%	2076	3	4	5	588
lunda	147	1151	[10772, 11323]	0,03%	11326	3	7	7	2483
lundb	147	1147	[10712, 11187]	0,04%	11192	3	7	7	2452
mesh33x33	1089	2112	31729	3,03%	32693	9	11	11	2764
nos4	100	247	**1031**	0%	1031	4	5	5	367
pores_1	30	103	**383**	0%	383	2	3	5	147
RandomA1	1000	4974	866968	3,00%	892986	10	10	10	26757
RandomA2	1000	24738	6522206	0,44%	6550805	10	10	10	196135
steam3	80	424	**1416**	0%	1416	4	4	5	842
tub100	100	148	**246**	0%	246	4	5	5	152
will57	57	127	**335**	0%	335	2	5	6	180

Table 4.2: Results for the minLA benchmark instances with 2 dimensions

Graph	$\|V\|$	$\|E\|$	**2D**					
			R	C	Cost	R	C	Cost
bcspwr01	39	46	3	13	57	5	8	51
bcspwr02	49	59	7	7	72		-	
bcspwr03	118	179	2	59	384	10	12	255
bcsstk01	48	176	6	8	384		-	
bcsstk02	66	2145	6	11	12155	8	9	11505
bcsstk04	132	1758	11	12	7126		-	
bcsstk05	153	1135	9	17	11060	11	14	3508
bcsstk22	110	254	10	11	374		-	
bintree10	1023	1022	31	33	1231	32	32	1233
c1y	828	1749	23	36	5760	27	31	5752
can___144	144	576	12	12	1058		-	
can___161	161	608	7	23	1371	12	14	1253
can___24	24	68	4	6	98		-	
can___61	61	248	1	61	1137	7	9	485
can___62	62	78	2	31	130	7	9	90
can___73	73	152	1	73	1100	7	11	284
can___96	96	336	8	12	600	9	11	599
curtis54	54	124	6	9	194	7	8	191
dwt___162	162	510	9	18	812	12	14	814
dwt___209	209	767	11	19	1588	14	15	1653
dwt___221	221	704	13	17	1184	15	15	1176
dwt___245	245	608	7	35	1143	15	17	1054
dwt___59	59	104	1	59	289	7	9	134
dwt___66	66	127	6	11	164	8	9	163
dwt___72	72	75	8	9	78		-	
dwt___87	87	227	3	29	448	8	11	384
fidap005	27	126	5	6	250		-	
fidapm05			6	7	545		-	
gd95c	62	144	2	31	318	7	9	233
gd96b	111	193	3	37	602	10	12	461
gd96c	65	125	5	13	196	8	9	188
gd96d	180	228	12	15	518	13	14	517
ibm32	32	90	4	8	192	5	7	183
impcol_b	59	281	1	59	2076	7	9	713
lunda	147	1151	7	21	2866	11	14	2802
lundb	147	1147	7	21	2836	11	14	2787
mesh33x33	1089	2112	33	33	2112		-	
nos4	100	247	10	10	424		-	
pores_1	30	103	5	6	167		-	
RandomA1	1000	4974	25	40	57855	29	35	57436
RandomA2	1000	24738	25	40	427480	29	35	415460
steam3	80	424	8	10	946		-	
tub100	100	148	10	10	158		-	
will57	57	127	3	19	218	7	9	187

Table 4.3: Results for the QAPlib benchmark instances with 1 & 3 dimensions

Graph	$\lvert V\rvert$	$\lvert E\rvert$	**1D**	**3D**			
			Cost	R	C	D	Cost
nug12	12	45	1000	2	2	3	524
nug14	14	68	1866	2	2	4	920
nug15	15	75	2186	2	2	4	1030
nug16a	16	93	3050	2	2	4	1398
nug16b	16	84	2400	2	2	4	1130
nug17	17	101	3388	2	3	3	1466
nug18	18	113	3986	2	3	3	1646
nug20	20	141	5642	2	2	5	2352
nug21	21	137	5084	2	3	4	1988
nug22	22	153	6184	2	3	4	2344
nug24	24	185	8270	2	3	4	2938
nug25	25	200	9236	3	3	3	3100
nug27	27	233	11768	3	3	3	3802
nug28	28	251	13090	2	3	5	4302
nug30	30	293	16502	2	3	5	5240
scr12	12	28	42776	2	2	3	30490
scr15	15	42	80862	2	2	4	49968
scr20	20	62	183270	2	2	5	101686
sko100a	100	3431	757188	4	5	5	103176
sko100b	100	3414	771792	4	5	5	104186
sko100c	100	3372	736510	4	5	5	100438
sko100d	100	3367	747542	4	5	5	101452
sko100e	100	3366	745104	4	5	5	101330
sko100f	100	3377	746562	4	5	5	100922
sko42	42	603	51050	2	3	7	13758
sko49	49	811	81964	2	5	5	18856
sko56	56	1061	128106	2	4	7	28396
sko64	64	1386	193878	4	4	4	34962
sko72	72	1781	278408	3	4	6	48800
sko81	81	2274	410562	3	3	9	73022
sko90	90	2771	547124	3	5	6	82248
ste36a	34	172	20574	3	3	4	8226
tho150	150	4732	48711062	5	5	6	5088332
tho30	30	217	348124	2	3	5	109408
tho40	40	312	729452	2	4	5	192988
wil100	100	4459	1372700	4	5	5	184756
wil50	50	1099	163508	2	5	5	37090

Table 4.4: Results for the QAPlib benchmark instances with 2 dimensions

Graph	$\|V\|$	$\|E\|$	**2D**							
			R	C	BKV	δ	Cost	R	C	Cost
nug12	12	45	3	4	**578**	0%	578	-		
nug14	14	68	3	5	**1014**	0%	1014	-		
nug15	15	75	3	5	**1150**	0%	1150	-		
nug16a	16	93	4	5	**1610**	0%	1550	4	4	1550
nug16b	16	84	4	4	**1240**	0%	1240	4	4	1240
nug17	17	101	4	5	**1732**	0%	1672	3	6	1672
nug18	18	113	4	5	**1930**	0%	1900	3	6	1900
nug20	20	141	4	5	**2570**	0%	2570	-		
nug21	21	137	3	7	**2438**	0%	2438	4	6	2270
nug22	22	153	2	11	**3596**	0%	3596	4	6	2742
nug24	24	185	4	6	**3488**	0%	3488	-		
nug25	25	200	5	5	**3744**	0%	3744	-		
nug27	27	233	3	9	**5234**	0%	5234	5	6	4612
nug28	28	251	4	7	**5166**	0%	5166	5	6	4988
nug30	30	293	5	6	**6124**	0%	6124	-		
scr12	12	28	3	4	**31410**	0%	31410	-		
scr15	15	42	4	4	**51140**	0%	51140	-		
scr20	20	62	5	4	**110030**	0%	110030	-		
sko100a	100	3431	10	10	152002	0,016%	152026	-		
sko100b	100	3414	10	10	153890	0,005%	153898	-		
sko100c	100	3372	10	10	147862	0%	147862	-		
sko100d	100	3367	10	10	149576	0,011%	149592	-		
sko100e	100	3366	10	10	149150	0,008%	149162	-		
sko100f	100	3377	10	10	149036	0,005%	149044	-		
sko42	42	603	6	7	15812	0%	15812	-		
sko49	49	811	7	7	23386	0%	23386	-		
sko56	56	1061	7	8	34458	0%	34458	-		
sko64	64	1386	8	8	48498	0%	48498	-		
sko72	72	1781	8	9	66256	0%	66256	-		
sko81	81	2274	9	9	90998	0%	90998	-		
sko90	90	2771	9	10	115534	0%	115534	-		
ste36a	34	172	2	17	**9526**	0%	9526	5	7	9258
tho150	150	4732	10	15	8133398	0,114%	8142732	12	13	7926106
tho30	30	217	3	10	**149936**	0%	149936	5	6	128772
tho40	40	312	4	10	240516	0%	240516	6	7	232752
wil100	100	4459	10	10	273038	0%	273038	-		
wil50	50	1099	5	10	48816	0%	48816	7	8	45672

Table 4.5: Parameters used depending on the number of edges of the graph

Parameter	**# Edges**				
	≤ 500	≤ 1000	≤ 5000	≤ 10000	≥ 10000
Iterations per Temperature	$2 \cdot 10^5$	$2 \cdot 10^6$	$2.5 \cdot 10^6$	$3 \cdot 10^6$	$3.5 \cdot 10^6$
T_{th}	0.25	0.5	0.75		
P_{th}	0.125	0.075		0.05	
Restart Factor γ	0.3	0.5	1	1.5	

4.3.1 Numerical Results

There are three types of best known values (BKV) that can be found in Tables 4.1, 4.2, 4.3 and 4.4. The first ones are optimal results (in boldface); the second ones are values computed by heuristics and, hence, we do not know whether they are optimal or not. Finally we have instances for which only upper and lower bounds are found in the literature and are represented with a range of values. The BKVs from Table 4.1 come from works [139] and [142] and the upper/lower bounds from [43] and [44]. On the other hand, the BKVs from Table 4.4 come from [55, 71, 119, 150]. The results shown in Tables 4.2 and 4.3, as well as the ones from the aforementioned tables which are not accompanied by a BKV value belong to new instances devised to work as new benchmark instances for MAP.

These results show that JAM is capable of matching most of the BKVs for the evaluated instances. Moreover, JAM even improved some of the results found in [44] for some minLA instances. The remarkable aspect of matching and improving some of these results is that, while they were achieved by heuristics devoted and optimized for a particular problem, JAM is able to perform with very competitive results with benchmark instances from multiple problems and in multiple dimensions. This fact also allows us to propose different layouts, enabling extra locations, that let us find layouts for which the evaluated graphs would reduce their costs. This means that, for an unconstrained real problem, we would be able to propose a layout with more locations than facilities and aim to find the best possible arrangement.

4.4 Conclusions

In this chapter we have presented the JAM algorithm for the Multidimensional Arrangement Problem. We have tested its practicality with benchmarks from the minLA and QAP literature. The results obtained with JAM often match the best known results and even improve some of them. Our experiments provide results for 1, 2 and 3 dimensions for 81 different graphs, broadening the available instances for both minLA and QAP as well as creating a valid set of benchmark instances for MAP.

Part III

Understanding and Reducing Energy Consumption in Data Centers

Chapter 5

Incorporating Rate Adaptation to Green Networking

5.1 Overview

This chapter is devoted to study and present some of the possible benefits that rate adaptation can bring to data center networking. Although we don't have the required technology available yet, some devices, like Infiniband devices [4], are already mimicking some of the aspects that are needed to use rate adaptation in Data Center Networks (DCN).

We start by presenting the energy-efficient routing problem we are dealing with. Then, making some realistic assumptions, we provide a formal model to describe how rate adaptation can be used in DCNs. However, we can easily find, analyzing the model, that our problem is NP-hard, in fact, it can be reduced to the edge-disjoint-path problem, a well known NP-hard problem.

This hardness comes from two different places, our non-convex objective function and a binary variable. Hence, we propose a two step relaxation process where we first study how to appropriately replace the step cost function with a polynomial cost function of the type μx^{α}; and, afterwards, replace our binary variable with a real one. Thanks to this two-step relaxation we are able to approximate the problem within a constant ratio to the optimum.

We will finally validate the performance of our algorithm by carrying out extensive simulations and finding how much energy savings can we get by applying rate adaptation to Data Center Networks. These simulations will be performed using real and synthetic traces as well as over different data center topologies, namely, a Fat Tree, BCube and Dcell.

RoadMap The remainder of this chapter is organized as follows. Section 5.2 models the rate-adaptive energy-efficient routing problem and analyzes the hardness of solving it. Section 5.3 presents our main algorithm and shows that the proposed algorithm can approximate the optimal solution within a constant factor. Section 5.4 verifies the performance of the proposed algorithm by simulations. Section 5.5 concludes the chapter.

5.2 Problem Description

We describe the rate-adaptive energy-efficient routing problem in detail in this section. We provide a formal model and analyze its complexity.

5.2.1 Basic Assumptions

We restrict our attention on communication intensive data centers such as MapReduce systems due to the following reasons: the network traffic in those systems is more considerable; because the energy consumption is also quite remarkable in these systems; and because it also more feasible to predict traffic with more accuracy. Also we do not claim any novelty and contribution on the system architecture and implementation as they can be simply adapted from ElasticTree [87]. Recall that in Elastic Tree, there are three main components: optimizer, power control and routing. The optimizer in Elastic Tree is responsible for finding network subset satisfying the current traffic conditions such that the power is minimized, while in our system it is replaced by solving the rate-adaptive energy-efficient routing problem, providing routes for flows and appropriate operating rates for network devices, as we will explain in detail later in this chapter. Similarly the power control in our system aims to control the operating rates of ports, links, and switches instead of toggling them on and off, while routing remains the same as in Elastic Tree.

5.2.2 Modeling

Now we describe the rate-adaptive energy-efficient routing problem that the optimizer aims at solving. Consider a data center network $G = (V, E)$, where V is the set of nodes and E is the collection of edges. Here nodes represent the chassises of network devices, while edges represent the network links and the corresponding link cards. In order to simplify the presentation, we consider the case where all the network devices used in a data center network are identical with the same technical specification, since data center networks such as FatTree, BCube are usually homogeneous. All the edges in E are assumed to have the capability of adapting their operating rates according to their carried traffic loads. It is quite reasonable that manufactures will provide many different operating rates in future network devices, due to the trend of being green. Assume we have given m discrete rates $R = (r_1, r_2, ..., r_m)$ for all the edges in E, where $r_i < r_{i+1}$ for $i \in [1, m-1]$. Each rate $r_i \in R$ $(1 \leq i \leq m)$ has a cost defined by function $f(r_i)$ which is uniform in the network, representing the power consumed by edge e when working at rate r_i.

We have a set of traffic demands $D = (d_1, d_2, ..., d_k)$ to be routed on G. For the i-th demand, a d_i units of bandwidth are requested to be provisioned from a source node s_i to a destination node t_i. For the simplicity of exposition, we first consider unit demands, i.e., $d_i = 1$ for $i \in [1, k]$. We will show later that the proposed algorithm can be adapted to other kinds of demands (uniform and non-uniform). In order to avoid packet reordering, we assume that all the demands will be routed in an unsplittable way, i.e., follow a single path (one TCP flow). The total cost of the network is defined as the summation of the costs of all the edges, which can be formalized as

$$C = \sum_{e \in E} C_e = \sum_{e \in E} f(z_e) \tag{5.1}$$

where z_e is the operating rate chosen for edge e. This cost represents the total power consumption of the whole network. The rate-adaptive energy-efficient routing problem now can be formulated with the following integer program.

$$(P_1) \quad \min \quad C = \textstyle\sum_{e \in E} f(z_e)$$

subject to

$$\begin{array}{ll} x_e = \sum_i \varphi_{i,e} & \forall e \\ x_e \leq z_e & \forall e \\ z_e \in \{r_1, r_2, ..., r_m\} & \forall e \\ \varphi_{i,e} \in \{0, 1\} & \forall i, e \\ \varphi_{i,e} : \text{ flow conservation} & \end{array}$$

where $\varphi_{i,e}$ is an indicator to show whether the i-th demand will be routed through link e which follows the flow conservation. Flow conservation means that for the i-th demand, the source s_i generates a flow of d_i (1 in this case) units and the sink absorbs it. For any other vertices, the ingress and egress flows are the same. x_e and z_e are the total load and the chosen operating rate for edge e respectively. For any rate $z_e \in \{r_1, ..., r_m\}$, we have $f(x_e) = f(z_e)$ if we choose z_e such that $x_e \leq z_e$. In other words, if we have $r_{j-1} < x_e \leq r_j$, then $f(x_e) = f(r_j)$, which inevitably results in the discreteness of the power cost function. In fact, $f(x)$ is a non-decreasing step function of the amount of the total traffic going through an edge.

Not surprisingly, solving P_1 is NP-hard, due to the fact that the objective function C is not convex. Moreover, it is also impossible to achieve any finite ratio approximation for this problem. This can be proved by a reduction from the edge-disjoint-path problem. Specifically, we can prove that any algorithm that gives a finite approximation ratio for (P_1) can be used to solve the edge-disjoint-path problem in polynomial time. This contradicts the fact that the edge-disjoint-path problem is NP-hard and have no polynomial time algorithms as long as P$\neq$NP. A detailed proof is provided in [163]. Observe that in this proof we use a step function $f(\cdot)$ with unbounded step ratios, which is exactly where the inapproximability comes from. Actually, a constant bound for step ratios can be assumed as the power consumption of network devices is usually bounded in reality. For this reason, we adopt this assumption in the rest of the chapter. Note that even so, the problem remains NP-hard and we have no hope on finding the optimal solutions. We aim at approaching efficient approximations.

5.3 Approximation

We present our main algorithm, a constant ratio approximation for solving the rate-adaptive energy-efficient routing problem. The main idea of this approximation algorithm is transforming program (P_1) into a standard convex program and solve it with convex programming, which is inspired by the observation that the complexity of the problem comes from the non-convexity of the cost function $f(\cdot)$. To this end, a two-step relaxation and rounding process is introduced in this algorithm. The algorithm solves the problem guaranteeing a constant approximation ratio. The details of the proofs of Theorems 9,10, and 11 belong to the work of Wang *et al.* and can be found in [163].

5.3.1 Relaxations

The first relaxation is made on the step function $f(\cdot)$. We try replacing $f(\cdot)$ with a convex function $g(\cdot)$ while introducing a bounded error. This transformation is accomplished by a special interpolation method.

Before introducing the interpolation, choosing an appropriate form for $g(\cdot)$ is essential as our goal is to determine the expression of $g(\cdot)$. Since function $f(\cdot)$ is the power curve of a network device, the target function $g(\cdot)$ should satisfy the basic properties of the way power being consumed by network devices. It has been suggested in [13] that most network components consume energy in a superadditive manner. Here we omit the idle power as the chassis of network devices are not considered in our model. Nevertheless, the idle power can be handled by the power-down based energy saving approaches which can be integrated with our model friendly. More formally, we set $g(\cdot)$ to be a polynomial function μx^α where μ and α are constant associated with the network devices. The constant α is usually assumed to be in $(1, 3]$ [40]. As a result, function $g(\cdot)$ will be convex and replacing $f(\cdot)$ with $g(\cdot)$ will simplify program (P_1).

Now the problem is how to determine parameters μ and α, with which the exact expression of $g(\cdot)$ can be obtained. We introduce an interpolation method for it. In order to contain the interpolation error, we devise a new interpolation method which aims at minimizing the difference between the two functions. As we have discussed in the previous section, function $f(\cdot)$ is defined as

$$f(x) = \begin{cases} p_1, & 1 \leq x \leq r_1, \\ p_2, & r_1 < x \leq r_2, \\ \ldots & \\ p_m, & r_{m-1} < x \leq r_m, \end{cases} \tag{5.2}$$

where $p_i = f(r_i)$ is the power consumption of an edge with operating rate r_i. r_i and r_{i+1} represent the lower and upper boundaries of each operating rate. Then, the interpolation is carried out by minimizing the maximal ratio between $f(\cdot)$ and $g(\cdot)$ which given by the following formula.

$$\Delta(\mu, \alpha) = \max_{x \in [1, r_m]} \left\{ \frac{f(x)}{g(x)}, \frac{g(x)}{f(x)} \right\}. \tag{5.3}$$

It is reasonable to restrict x to interval $[1, r_m]$ as $f(0) = g(0)$ and in any feasible integral solution, $x \notin (0, 1)$. As $g(\cdot)$ is not linear, this minimization problem is hard to tackle. But it can be observed that $g(\cdot)$ becomes linear if we apply a logarithmic transformation on it, as well as $\Delta(\mu, \alpha)$. Then, it is equivalent to minimize

$$\begin{aligned} \Phi(\mu, \alpha) &= \max_{x \in [1, r_m]} |\log f(x) - \log g(x)| && (5.4) \\ &= \max_{x \in [1, r_m]} |\log f(x) - (\log \mu + \alpha \log x)| && (5.5) \end{aligned}$$

Nevertheless, the absolute operation becomes an obstacle. We propose to use an alternative, minimizing the integral of the square of the difference between the two functions. Let $v = \log x$ and $q = \log f(x)$. Then, we have $v_i = \log r_i$ and $q_i = \log p_i$ for $1 \leq i \leq m$. We set $v_0 = \log 1 = 0$ and $\lambda = \log \mu$. Then, the alternative we aim to minimize is

$$\Psi(\lambda, \alpha) = \sum_{i=1}^{m} \int_{v_{i-1}}^{v_i} (q_i - (\lambda + \alpha v))^2 \, dv. \tag{5.6}$$

The parameters λ and α can then be obtained by minimizing the above formula, which can be achieved by setting the first derivatives of $\Psi(\lambda, \alpha)$ with respect to λ and α to zero. This is due to the fact that minimizing $\Psi(\lambda, \alpha)$ is a quadratic optimization problem and its second derivatives are all positive.

We define the error in this interpolation as the maximum ratio between $f(\cdot)$ and $g(\cdot)$, i.e., $\Delta(\mu, \alpha)$. We also assume that $g(\cdot)$ intersects with $f(\cdot)$ in every step of $f(\cdot)$, which is quite reasonable because $g(\cdot)$ is assumed to be a proper description of the superadditive fashion power being consumed. If not, there could be a big difference in trend between $f(\cdot)$ and $g(\cdot)$ and we cannot obtain any bounds. In other words, this would mean that the superadditive rule doesn't apply to this network. Combining with our another assumption that the ratio of the steps of $f(\cdot)$ is bounded by a constant, we have the following result.

Theorem 9. *Given function $f(x)$, if $f(x)$ satisfies $p_i/p_{i-1} \leq \delta$ where $\delta > 1$, then in interval $[1, r_m]$, the interpolation error satisfies*

$$\frac{\delta}{\delta + 1} \leq \Delta(\mu, \alpha) \leq \frac{\max\{\delta, f(1)\}}{\mu}. \tag{5.7}$$

The proof is conducted by considering two cases $f(x) \geq g(x)$ and $f(x) < g(x)$. In both cases we assume there is a bound for the interpolation error, and then we derive the conditions that this bound needs to satisfy in order to maintain the bound.

With function $g(\cdot)$ determined, a new program is then obtained by replacing $f(\cdot)$ with $g(\cdot)$ in the objective of (P_1).

$$(P_2) \quad \min \quad \bar{C} = \textstyle\sum_{e \in E} g(x_e)$$

subject to

$$\begin{array}{ll} x_e = \sum_i \varphi_{i,e} & \forall e \\ x_e \leq r_m & \forall e \\ \varphi_{i,e} \in \{0,1\} & \forall i, e \\ \varphi_{i,e} : \text{ flow conservation} & \end{array}$$

where the second constraint means that no edge can be loaded beyond its maximum operating rate. However, solving (P_2) is still NP-hard because of the binary constraint.

Note that (P_2) is a convex program with binary variable $\varphi_{i,e}$. It is then obvious that (P_2) can be optimally solved if we relax the binary variable to real values. By replacing constraint $\varphi_{i,e} \in \{0,1\}$ with $\varphi_{i,e} \in [0,1]$, the resulted program can be efficiently solved by convex programming algorithms. We denote the solution obtained by convex programming, the optimal fractional solution, as $\varphi^*_{i,e}$. Notice that in this solution, a demand can be splitted over multiple paths, which is not feasible to our problem. We will show how to transform the optimal fractional solution to a feasible one to (P_1) by rounding operations.

5.3.2 Roundings

We introduce now some rounding techniques in order to retrieve feasibility from the optimal fractional solution. The rounding process contains two main steps which correspond to the two steps in the relaxation process. We will show that a feasible solution that is within a constant factor of the optimal can be obtained for the original problem by applying the proposed two-step rounding process.

The first rounding step is performed to convert the routes for demands from multi-path to single-path. Recall that in the optimal fractional solution $\varphi^*_{i,e}$, we may have more than one path for routing each demand. Our goal is to choose a single path P_i for each demand d_i such that $\{e \mid e \in P_i\} \subseteq \{e \mid \varphi^*_{i,e} > 0\}$. To this end, we borrow here the Raghavan-Thompson randomized rounding technique. The overall rounding procedure is described as follows: once the optimal fractional solution $\varphi^*_{i,e}$ has been found, the flows assigned to edges are mapped to paths. For each demand d_i, we construct a graph G_i using the edge that satisfy $\varphi^*_{i,e} > 0$ and their corresponding nodes. Then, we extract a simple path A connecting the source and destination nodes. The edge e that has the smallest $\varphi^*_{i,e}$ in this path will be chosen as the bottleneck edge and the corresponding $\varphi^*_{i,e}$ is selected as the weight of this path, denoted as W_A. After that, the weight of every edge on this path will be decreased by W_A, i.e., $W_e \leftarrow W_e - W_A$ for all $e \in A$. The above path extraction operation will be repeated until $\varphi_{i,e} = 0$ for all $e \in E$. Note that this will always happen due to the flow conservation constraint. As a result, for this demand, we have obtained a collection of paths $\{A\}$ attached with weights $\{W_A\}$. We then randomly select one path from all the candidate paths $\{A\}$ using the path weights as the selection probabilities. This selected path will be used as the route for demand d_i. When all the demands have been processed, we denote the obtained solution as $\hat{\varphi}_{i,e}$.

The operating rate of each edge can then be determined according to the load each edge carries which is calculated by $\hat{x}_e = \sum_{i\in[1,m]} \hat{\varphi}_{i,e}$. Then, the minimum speed in R that can support this load is selected for this edge to operate. Formally, we choose

$$z_e = \min\{r_i \in R \mid i \in [1,m] \wedge \hat{x}_e \leq r_i\}. \tag{5.8}$$

Denote the solution obtained after this rounding procedure as $\varphi_{i,e}$, which is feasible for our original problem (P_1). At the same time, we have the following theorem.

Theorem 10. *For unit demands, the expected power consumption obtained by the two-step relaxation and rounding algorithm is a γ-approximation of the expected optimal power consumption, where γ is a constant that depends on σ and δ.*

The proof of this theorem can be conducted by analyzing the error introduced in each step and combining them together. We omit the details here. This result can also be extended to cases with uniform and non-uniform demands. Specifically, we have

Corollary 13. *The two-step relaxation and rounding algorithm can guarantee a γ-approximation for the rate-adaptive energy-efficient routing problem with uniform demands.*

Theorem 11. *For non-uniform demands, the two-step relaxation and rounding algorithm can achieve a $O(\log^{\alpha-1} d)$-approximation for the rate-adaptive energy-efficient routing problem, where $d = \max_{i\in[1,k]} d_i$.*

For the case with uniform demands, Theorem 10 can be easily extended by normalizing all the demands to unit demands, while the case with non-uniform demands needs some results from [13].

5.4 Evaluation

We carry out comprehensive simulations to evaluate the performance of our proposed algorithm in this section. Particularly, we will focus on four main aspects: interpolation error, approximation ratio, potential energy saving effect and delay stretch.

5.4.1 Experimental Settings

We use a synthetic power function $f(\cdot)$ which we believe is similar to the fashion power being consumed by future rate-adaptive commodity network devices, as shown in Figure 5.1. The maximum operating rate of network devices is set to be 60 units. We also believe that the precise

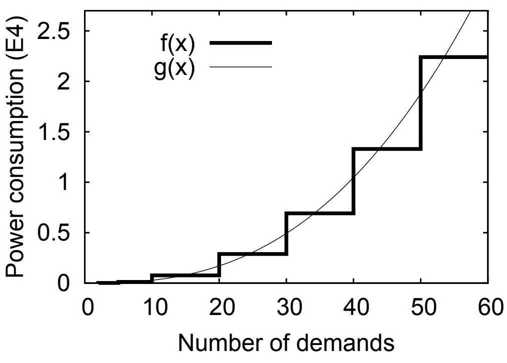


Figure 5.1: The synthetic power function $f(\cdot)$ and the convex alternative $g(\cdot)$ obtained by the proposed interpolation.

Table 5.1: Main parameters for the three topologies

Topology	**# of hosts**	**# of switches**	**# of links**
FatTree	128	80	384
BCube	144	24	288
DCell	132	12	198

form of $f(\cdot)$ can be easily obtained from vendors in the future. The alternative convex function $g(\cdot)$ is obtained by applying the interpolation method we proposed in section 5.3.1, as also shown in Figure 5.1.

We carry out the validation for both interpolation error and approximation ratio basing on two topologies in different scales, 4-ary and 20-ary FatTree. The number of demands are varied from on to six times the number of physical machines in each topology. The source and destination nodes of these demands are chosen uniformly at random from these physical machines, while the bandwidth each demand requests is generated randomly following a normal distribution $\mathcal{N}(1, 0.2)$.

The efficiency of energy saving of our algorithm is verified with both synthetic traffic conditions and real network traces. In both cases, the numbers of physical machines are chosen as 128 which is determined by the real network traces. We conduct our simulations using three popular kinds of data center network topology: FatTree, BCube and DCell. The detailed parameters of the three topologies are demonstrated in Table 5.1. These parameters are determined by the topology characteristics and our requirement that at least 128 physical machines have to be involved. The traffic condition for the synthetic testing is generated using the same setting as described before, while for the real testing it is extracted from a university data center [35]. The traffic patterns of the real network traces is worth 15 minutes long and the endpoints in the traces are mapped to the three topologies we use.

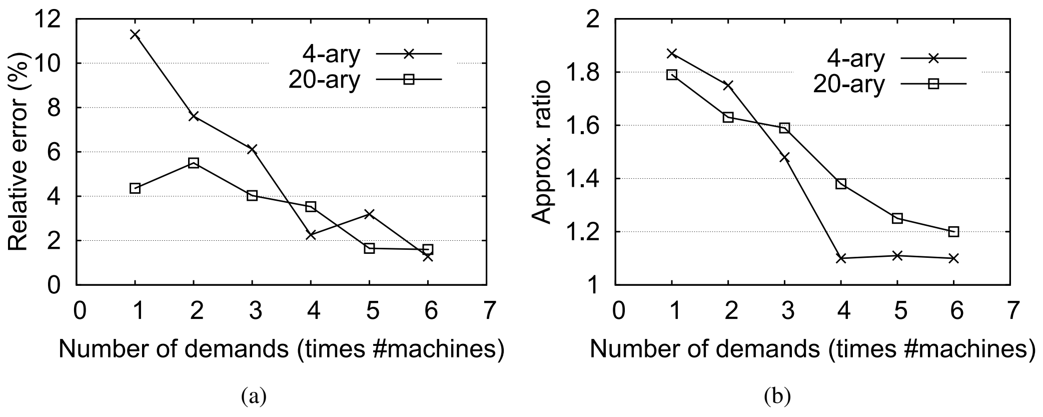


Figure 5.2: Simulation results for testing (a) the interpolation error and (b) the approximation ratio of the proposed algorithm based on two topologies: 4-ary and 20-ary FatTree.

5.4.2 Interpolation Error

We verify the interpolation error in this subsection since the effectiveness of the interpolation will condition heavily the whole algorithm. Although we have already proved that this error is bounded by some constant, the simulation results will help us know that the error occurred in the interpolation is also ignorable in practice. Here we define the relative error as

$$\frac{|E_g - E_f|}{E_f} \times 100\%, \tag{5.9}$$

where E_f is the power consumption under $f(\cdot)$ while E_g is the power consumed under $g(\cdot)$. As we cannot solve the routing problem under $f(\cdot)$ efficiently, we propose to use the shortest-path routing algorithm instead. This is reasonable because the interpolation does not reply on the routing method. The shortest path routing is accomplished by Dijkstra algorithm. We calculate E_f and E_g with power curve $f(\cdot)$ and $g(\cdot)$ respectively when routing demands in shortest path manner.

The simulation results are shown in Figure 5.2(a), where all the values are averaged among ten independent repeats. We can observe that the interpolation error is quite small in both small- and large-scale networks, while even smaller in the large-scale network. With the number of demands increasing, this error decreases dramatically. When the number of demands is large enough (e.g. four times the number of physical machines), the interpolation error is within 4% and is around 2% during most of the time. Considering the fact that each physical machine that is involved in a MapReduce job has to communicate with a set of other physical machines for the same job, having the number of demands more than four times the number of physical machines is quite reasonable in communication-intensive systems. This convinces us that the proposed interpolation method is efficient enough in practice.

5.4.3 Approximation Ratio

The theoretical analysis has given a strict constant bound for the approximation ratio. However, it is still necessary to understand the goodness when the constant approximation performs in real data centers. We verify now the approximation efficiency by simulations. Basing on two topologies of FatTree in different scales, we compare our solution with the optimal fractional solution as the optimal fractional solution is a lower bound for the optimal integral solution. The obtained ratio between our solution and the fractional optimal solution will be an upper bound for the approximation ratio.

The simulation results are presented in Figure 5.2(b) where all the values are averaged among ten independent repeats. It can be noticed that the approximation ratio of the proposed algorithm is quite small and approximately converges to 1 quickly with the increase of the number of demands, regardless of the scale of the topology. This confirms that the constant approximation performs nearly optimally in large-scale data center networks.

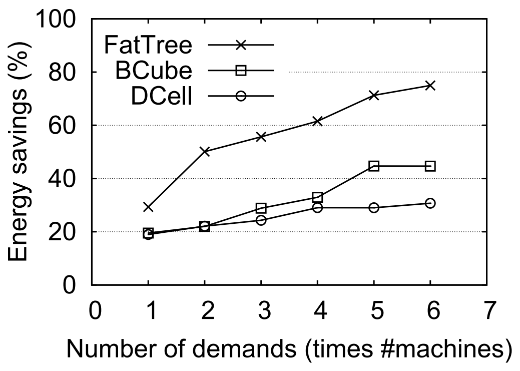
5.4.4 Potential Energy Savings

We evaluate now the energy saving efficiency of the proposed routing algorithm when being applied in real data centers. To be fair, we assume that all the network devices are capable of rate adaptation no matter what routing algorithm is used. We compare our algorithm with the shortest path based routing which is a common practice in most networks. We focus on two aspects of interest - the energy savings and the ratio of idle links (not used for routing any demand). The energy saving ratio is calculated as the ratio between the energy consumption of our algorithm and the shortest path based algorithm, while the ratio of idle links is calculated using the number of idle links divided by the total number of links.

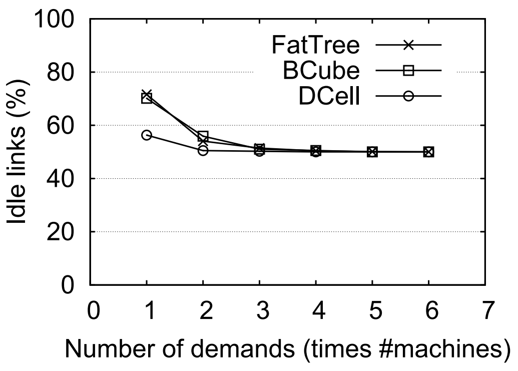
5.4.4.1 Synthetic Traffic

The simulation results under synthetic network traffic are shown in Figure 5.3. It can be observed that the proposed energy-efficient routing approximation algorithm can achieve up to 60% energy savings in FatTree and more than 30% in both BCube and DCell. Moreover, this savings tends to be stable with the increase of the number of demands, convincing us that a global energy-efficient routing optimization can help reduce a substantial amount of power consumption.

We also observe that the link utilization in a data center stays quite low during most of the time. As shown in Fig. 5.3(b), with a reasonable amount of demands, only 50% of the links are needed. This reveals that for normal traffic patterns, all the traffic can be carried by only half of the total links, as a consequence of the high link redundancy in the network.



(a)



(b)

Figure 5.3: Simulation results for testing the energy saving efficiency using synthetic network conditions based on three different topologies: FatTree, BCube and DCell.

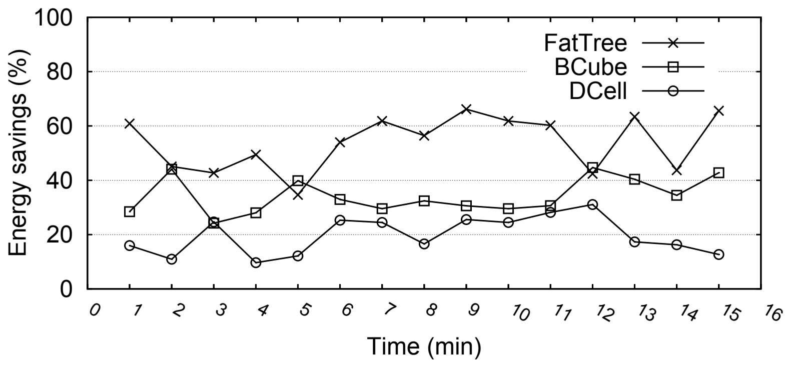
5.4.4.2 Real Traces

We repeat the above experiments using real traces from a university data center. The numerical results are shown in Fig. 5.4. We can observe that the average energy saving is more than 50% in FatTree, while this value is 30% and 20% in BCube and DCell respectively. This is because the link redundancy is more significant in FatTree topologies, leading to better efficiency of the network-global routing optimization. At the same time, the ratios of idle links presented in Fig. 5.4(b) also proves that we can use about half of the links to carry all the traffic on a real data center network.

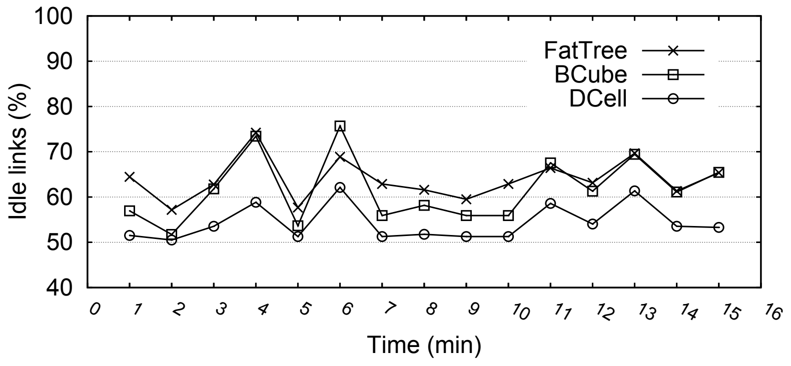
Recall that in our model we do not have any assumption on powering down network devices. Meanwhile, the rounding process in Equation 5.8 provides some extra capacity for each edge, leading to better stability in the network while compared with power-down based approaches. Nevertheless, we claim that our proposed method can also be integrated with power-down based approaches for the reason that the ratio of idle links in data center networks is usually quite high as indicated above. As a result, further energy can be saved by switching off these unused links.

5.4.5 Delay Stretch

We now focus on a main aspect of network performance, the network delay. It is quite possible that the proposed energy-efficient routing will use more hops to route a demand than the shortest path based routing, bringing about lager network delay, thus the degradation of the network quality of services. In order to justify how bad it can be, we compare the average hops for routing each demand with the two routing algorithms. This comparison is quite sensible since the maximum number of hops is bounded by the topology diameter in most data center networks. The results are illustrated in Fig. 5.5(b). We can notice that the overhead brought by the energy-efficient routing is always within one hop, being acceptable then. Furthermore, there is no delay overhead in FatTree topology due to the special hierarchical property of FatTree. As a result,

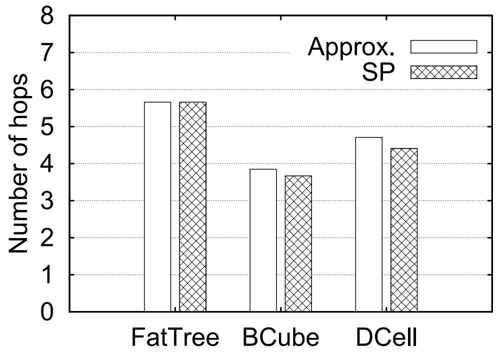


(a)

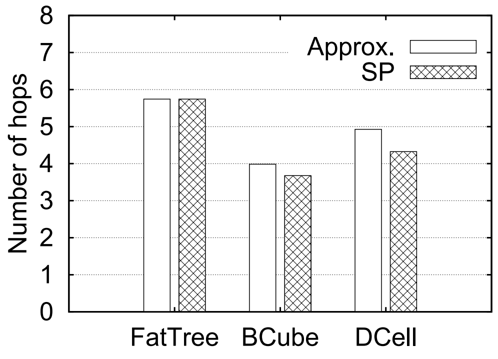


(b)

Figure 5.4: Simulation results for testing the energy saving efficiency using real network traces based on three different topologies: FatTree, BCube and DCell.



(a)



(b)

Figure 5.5: The average number of hops for routing each demand under (a) the synthetic traffic, (b) the real network traces.

it is evident that the energy-efficient routing is capable of reducing energy consumption while introducing very small stretch to the network delay.

5.5 Conclusions

In this chapter, we have shown the potential of saving energy by introducing rate adaptation in data center networks. Compared with power-down based approaches, rate adaptation is advantageous because of the better stability when being applied in networks. It has been shown that network-global optimization is necessary in order to achieve energy proportionality for the whole network with energy-proportional network devices. However, it is also known that this network-global optimization problem is usually hard to solve. To this end, we devise an approximation algorithm basing on a two-step relaxation and rounding process. The proposed algorithm performs very well by obtaining nearly optimal solutions in practice, while guaranteeing a constant approximation ratio in theory. Comprehensive simulations show that by incorporating rate adaptation into data center networks, the network-global routing optimization can bring up to 40% energy savings, even without switching off any network devices.

Chapter 6

Analysis of the Energy Consumption of Data Center Servers

6.1 Overview

The work presented in this chapter is motivated by our disagreement with some of the models that have been previously proposed in the literature which state that power consumption of data center servers depends linearly on the load. Our belief is that more complex/complete models for the power consumed by a server are necessary. In order to be consistent, these models have to be based on empirical values. However, we found that, despite the large body of work in the field, there is a lack of empirical work studying servers energy behavior.

Our work tries to partially fill this void by proposing a measurement-based characterization — which is the first of its kind— of the energy consumption of a server components with DVFS and multiple cores. We evaluate here different server machines and evaluate what is the contribution to their power consumption of the CPU, hard drive disk, and network card (NIC). Our approach captures the influence of the processing frequency and the multiple cores, not only to the CPU power consumption, but also to that of disk input/output (I/O) and NIC activity.

Our contribution is threefold: (i) we propose a methodology to empirically characterize the energy consumption of a server, (ii) we provide novel, experimental-based, insights on the energy consumption behavior of the most relevant server's components, and (iii) we propose an accurate technique to estimate the energy consumption of cloud applications.

As concerns the methodology, we propose *active CPU cycles per second* (ACPS) as a new and more convenient metric for CPU load in multi-core/multi-frequency architectures. We show how to isolate the contribution of energy consumption due to CPU, disk I/O operations, and network activity by just measuring server's total energy consumption and a few activity indicators reported by the operating system. We also show that the *baseline* energy consumption of a server — i.e., the energy consumed just because the server is turned on — has a strong impact on server's total consumption.

As concerns the components' energy characterization, we show that, besides the *baseline* consumption, the CPU has the largest impact among all components, and its energy consumption is not linear with the load. Disk I/O operations are the second highest cause of consumption, and their efficiency is strongly affected by the I/O block size used by the application. Eventually, network activity plays a minor yet not negligible role in the energy consumption, and the network impact scales almost linearly with the network transmission rate. All other components (e.g., memory, fans, GPU, etc.) can be accounted for the *baseline* energy consumption, which is subject to minor variations under different operational conditions. Specifically, the main results of our measurement campaign are listed below:

- The CPU power utilization depends on the number of working cores, the CPU frequency, and the CPU load (in ACPS units). Our measurements confirm that the energy consumption with a single working core at constant frequency can be closely approximated by a linear function of the CPU load. However, given a CPU frequency, the energy consumption in multicore architectures is a concave function of the CPU load and can be approximated by a low-order polynomial. The energy consumption for a fixed CPU load is, in general, minimized by using the highest number of cores and the lowest frequency at which the load can be served. However, the minimum achievable energy consumption is a piecewise concave function of the CPU load.

- The energy consumed by hard disks for reading and writing depends on the CPU frequency and the I/O block sizes. Both reading and writing energy costs increase slightly with the CPU frequency. While the energy consumption due to reading is not affected by block size, the energy consumption due to writing increases with the block size. The reading efficiency (expressed in MB/J) is barely affected by the CPU frequency, while writing efficiency is a concave function of the block size since it boosts the throughput of writing until a saturation value is reached.

- The energy consumption and the efficiency of the NIC, both in transmission and reception, depends on the CPU frequency, the packet size, and the transmission rate. The efficiency of data transmission increases almost linearly with the transmission rate, with steeper slopes corresponding to lower CPU frequencies. Although a linear relation between transmission rate and efficiency holds for data reception as well, small packet sizes yield higher efficiency in reception.

Overall, supported by our measurements, we provide a holistic energy consumption model that only requires a few calibration parameters for every different server architecture which we want to evaluate (a universal energy model will be too simplistic and inaccurate). We validate our model by means of a server computing the *PageRank* metric of a graph and a *WordCount* application in a *Hadoop* platform, first without network activity, next with bulky network activity, and finally in the cloud. We will find that the error of our energy estimates is below 4.1% on average and never worse than a 10%.

RoadMap The rest of the chapter is organized as follows. Section 6.2 describes the methodology we used for our experiments. Section 6.3 presents our measurement campaign, for every single component which we tested. In Section 6.4 we model the energy consumption of the servers based on a few calibration parameters which we find during our measurement campaign. In Section 6.5 we discuss our findings and their implications. Finally, Section 6.6 concludes the chapter.

6.2 Methodology

In this section we introduce the measurement techniques we used to characterize the power requirements of CPU activity, disk access (read and write operations), and network activity. Our measurements start characterizing the CPU power consumption, from where we obtain information about the baseline power consumption of the system. After CPU and baseline characterization, we follow with experiments for the other two components, namely, disk and network. Note that CPU and baseline measurements are of capital importance in order to evaluate the other components, because any operation run in a machine is like a puzzle with multiple pieces and we must know what is the contribution of each one of these pieces. Consider that, we are paying a cost just for having a server switched on and the operating system running on it. Similarly, every time we run a task in the system, some CPU cycles are needed in order to execute it as well as to use the component that has to perform the task. Hence, in order to understand the contribution of any component, we first need to identify the contribution of the CPU and compute the difference with respect to the aforementioned baseline.

To explore the possible parameters which determine the power consumption of a server and to obtain statistical consistency, we run our experiments multiple times. Similarly, we run these experiments in different servers and architectures in order to validate our results and give consistency to our conclusions.

6.2.1 Collecting System Data and Fixing Frequency Parameters

One prerequisite for our experiments is to have Linux machines due to the kind of commands and benchmarks we wanted to use and, mainly, because of the possibility of adding some kernel modules and utilities,[1] which allows us to change CPU frequencies at will. In a Linux system, CPU activity stats are constantly logged, so we can periodically record the core frequency and the number of *active* and *passive* CPU ticks at each core.[2] Once we have the number of ticks and

[1]For instance cpufrequtils, acpi-cpufreq.

[2]File `/proc/stat` reports the number of ticks since the computer started devoted to *user*, *niced* and *system* processes, waiting (*iowait*), processing interrupts (i.e., *irq* and *softirq*), and *idle*. In our experiments we count both

the core frequency, since a tick represents a hundredth of second, cycles can be calculated as 100 *ticks/frequency*.

We use active cycles per second (ACPS) instead of CPU load percentage to characterize CPU load. ACPS on the CPU frequency used, as the higher the frequency the more the work that can be processed. Hence, a percentage of load is not comparable when different frequencies are used, while the amount of ACPS that can be processed can be considered as an absolute magnitude. In order to get (set) information about the operative frequency of the system we used the `cpufrequtils` package.[3] With those tools, we can monitor the CPU frequency at which the system works and assign different frequencies to the cores. However, to limit the number of possible combinations to characterize, we assign the same frequency to all cores.

6.2.2 CPU

In order to evaluate the CPU power requirements we prepared a script based on a benchmark application, namely `lookbusy`.[4] Note that `lookbusy` allows us to load one or more CPU cores with the same load.Our `lookbusy`-based experiment follows the next steps: we first fix the CPU frequency to the lowest possible frequency in the system; then we run `lookbusy` with fixed amount of load for one core during timeslots of 30 seconds, starting with the maximum load and then decreasing the load gradually. After the last `lookbusy` run we measure the power required during an additional timeslot with *no* `lookbusy` load offered. We register the active cycles and the power used during each timeslot.

After taking these different samples for one frequency we move to the immediately higher frequency (we can list and change frequencies thanks to `cpufrequtils`) and repeat the previous steps. After going through all the available frequencies, we restart the whole process but increasing by one the number of active cores. We repeat this whole process until all the cores of the server are active. Note that when we change the frequency of the cores we change it in all of them, active or not, for consistency. Similarly, when we have more than one active core, the load for all the active cores is the same.

Once explained the scheme of our experiments, we must clarify the meaning of running a timeslot with *no* load. Note that zero-load is clearly not possible as there is always going to be load in the system due to, e.g., the operating system. However, during the timeslot in which we do not run `lookbusy`, we measure the power corresponding to the operational conditions which are as close as possible to the ones of an idle system. Moreover, the decision of using timeslots of 30 seconds is to guarantee enough, yet not excessive, time for the measurements. In fact, as we start and stop `lookbusy` at the beginning and end of the timeslots, we need to ignore the first and the last few seconds of measurements in each timeslot to avoid measurement noise due to power ramps and operational transitions.

waiting and idle ticks as *passive* ticks, while we denote the aggregated value of the rest of ticks as *active*.

[3] `https://wiki.archlinux.org/index.php/CPU_Frequency_Scaling`

[4] `http://www.devin.com/lookbusy.`

The measured values of load (in ACPS) and power in each timeslot are used to obtain a least squares polynomial fittings curve. These fittings characterize the CPU power requirements for each combination of frequency and number of active cores. We will use as *baseline power consumption* of each one of these configurations the zero-order coefficient of the polynomial of these fittings curves.

6.2.3 Disks

The power consumption of the hard drive was evaluated using 2 different scripts (for reading and writing) based on the `dd` linux command.[5] We chose `dd` as it allows us to read files, write files from scratch, control the size of the blocks we write (read), control the amount of blocks written (read) and force the commit of writing operations after each block in order to reduce the effect of operating system caches and memory. We combine this tool with flushing the RAM and caches after each reading experiment.

In both our scripts we perform write (read) operations for a set of different I/O block sizes and for different data volumes to be written (read). In each case we record the CPU active cycles, the total power and time consumed in each one of these operations for each combination of block size and available frequency.

Finally, we identify the contribution of the hard drive to the total power required by subtracting the contribution of both the baseline and the CPU requirements from the measured total power.

Disk I/O experiments shed light on the relevance of the block sizes when reading or writing as well as whether there is an influence of the frequency on these operations.

6.2.4 Network

In order to evaluate the contribution of the network to the power requirements of a server, we devised a set of experiments based on a client-server C script devised on purpose for this task.

There are several aspects that we consider relevant in order to characterize the impact of the NIC on the total power requirements of a server and that led us to choose these two tools. First, the ability of performing tests in which the server under study acts as sender or as receiver during a network connection, and therefore we can observe server's power requirements while sending data or receiving it. To clarify the terms, *sender* is the server which injects traffic to the network, and *receiver* is the server which accepts traffic from the network. Second, the ability of those tools to change several parameters that we consider relevant for the energy characterization of the servers, namely, the packet size and the offered load, jointly with the frequency of the system.

Our experiments consist, then, on measuring the achieved data rate, the CPU active cycles per second (ACPS) and the total power required by the server under study either as sender or as receiver using different packet sizes and different transfer rates. We run each experiment multiple times for statistical consistency.

[5]`http://linux.die.net/man/1/dd`.

Table 6.1: Characteristics of the servers under study

Component	Servers		
	`Survivor`	`Nemesis`	`Erdos`
CPU (# cores)	4	4	64
# freqs	8	11	5
Freqs List (*GHz*)	1.2, 1.333, 1.467, 1.6, 1.733, 1.867, 2, 2.133	1.596, 1.729, 1.862, 1.995, 2.128, 2.261, 2.394, 2.527, 2.666, 2.793, 2.794	1.4, 1.6, 1.8, 2.1, 2.3
RAM	4 *GB*	4 *GB*	512 *GB*
Disk	2 *TB*	$2+3$ *TB*	2×146*GB* 4×1 *TB*
Network	1 *Gbps*	3×1 *Gbps*	4×1 *Gbps* 2×10 *Gbps*
Architecture	Intel	Intel	AMD

Finally, using the CPU active cycles per second which were measured during the experiment, we identify the power required by the CPU. Subtracting both CPU power requirements and the baseline power from the total energy consumption of the experiment, we can isolate the power requirements of the network.

6.3 Measurements

6.3.1 Devices and Setup

In order to monitor and store the instantaneous power required by a server during the different experiments we used a Voltech PM1000+ power analyzer[6], which is able to measure the total instantaneous power required by the server under test on a per-second basis. In order to take our measurements we connected the server being measured to the power analyzer and the latter to the power supply. In the experiments where the network was not involved (CPU and disk), we disconnected the server from the network, which has an impact on the power requirements as the port goes idle. In the network based experiments we established an Ethernet connection between the server under study and a second machine in order to study the server behavior, both as receiver as well as as sender.

We evaluated three different servers: `Survivor`, `Nemesis`, and `Erdos`. We will now present these servers although their main characteristics, including their sets of available CPU frequencies, can be also found in Table 6.1. `Survivor` has an Intel Xeon E5606 4-core processor, with 4 *GB* of RAM, a 2 *TB* Seagate Barracuda XT hard drive and a 1 *Gigabit* Ethernet card integrated in the motherboard. `Nemesis` is a Dell Precision T3500 with an Intel Xeon W3530 4-core processor, 4 *GB* of RAM, 2 hard drives (a 2 *TB* Seagate Barracuda XT and a 3 *TB* Seagate Barracuda), a 1 *Gigabit* Ethernet card integrated in the motherboard, and a separate Ethernet card

[6]`http://www.farnell.com/datasheets/320316.pdf`

with two 1 *Gigabit* ports. In this study we only evaluate the Seagate Barracuda XT disk and the integrated Ethernet card. Both `Survivor` and `Nemesis` use the Ubuntu Server edition 10.4 LTS Linux distribution. Finally, `Erdos` is a Dell PowerEdge R815 with 4 AMD Opteron 6276 16-core processors (i.e., 64 cores in total), 512 *GB* of RAM, two 146 *GB* SAS hard drives configured as a single RAID1 system (which is the "disk" analyzed here) and four 1 *TB* Near-line SAS hard drives. It also includes four 1 *Gigabit* and two 10 *Gigabit* ports. `Erdos` is a high-end server and uses Linux Debian 7 Wheezy.

6.3.2 Baseline and CPU

As mentioned in the previous section, for each server we have measured the power it consumes without disk accesses nor network traffic. We assume that the power consumption observed is the sum of the baseline consumption plus the power consumed by the CPU. We have obtained samples of the power consumed under different configurations that vary in the number of active cores used, the frequency at which the CPU operates (all cores operate at the same frequency), and the load of the active cores (all active cores are equally loaded). The list of available and tested CPU frequencies and cores can be found in Table 6.1. We tune the total load ρ by using `lookbusy`, as described in the previous section. Each experiment lasts 30 s and it is repeated 10 times. Results are summarized in terms of average and standard deviation. Specifically, in the figures reported in this section, the power consumption for each tested configuration is depicted by means of a vertical segment centered on the average power consumption measured, and with segment size equal to two times the standard deviation of the samples.

The results of these experiments for each of the 3 servers are presented in Figure 6.1 (the measurements for some frequencies and some number of cores are omitted for clarity). Here, for each configuration of number of active cores, frequency, and load in ACPS, the mean and standard deviation of all the experiments with that configuration are presented. Also the least squares polynomial fitting curve for the samples is shown for each number of cores and frequency. The curves shown are for polynomials of degree 7, but we observed that using a degree 3 polynomial instead does not reduce drastically the quality of the fit (e.g., the relative average error of the fitting increases from 0.7% with 7-th degree polynomials to 1.5% with degree equal to 3 for `Erdos`, while it remains practically stable and below 0.7% for `Nemesis`). In general, we can use an expression like the following to characterize the CPU power consumption:

$$P_{BC}(\rho) = \sum_{k=0}^{n} \alpha_k \rho^k, \quad n \leq 7, \tag{6.1}$$

where P_{BC} includes both the baseline power consumption of the servers and the power consumed by the CPU, and ρ is the load expressed in active cycles per second. Therefore, coefficient α_0 in Eq. 6.1 represents the consumption of the system when the CPU activity tends to 0, and we can thereby interpret α_0 as the baseline power consumption of the system. Note that the polynomial fitting, and hence the baseline power consumption α_0, depends on the particular combination of number of cores and frequency adopted. However, for sake of readability, we do not explicitly account for such a dependency in the notation.

A first observation of the fitting curves for each particular server in Figure 6.1 reveals that the power for near-zero load is almost the same in curves (e.g., for `Nemesis` this value is between 84 and 85 W). Observe that it is impossible to run an experiment in which the load of the CPU is actually zero to obtain the baseline power consumption of a server. However, all the fitting curves converge to a similar value for $\rho \to 0$, which can be assumed to represent the baseline power consumption.

A second observation is that for one core the curves grow linearly with the load. However, as soon as two or more cores are used, the curves are clearly concave, which implies that for a fixed frequency the efficiency grows with the load (we will discuss later the efficiency in terms of number of active cycles per energy unit).

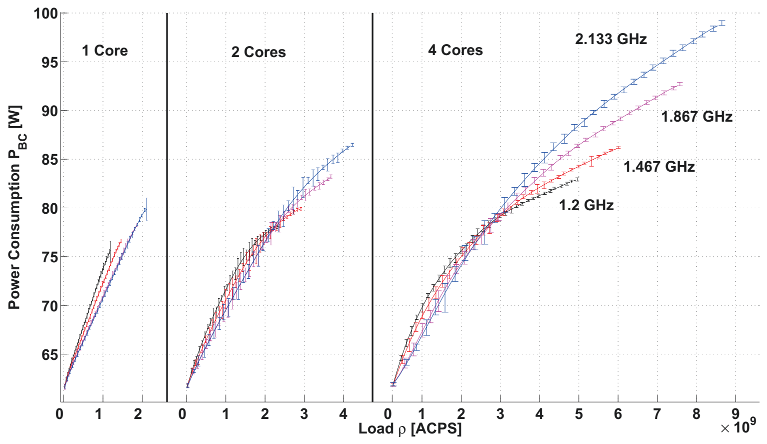
A third observation is that frequency does not significantly impact the power consumption when the load is low. In contrast, at high load, the consumption clearly increases with the CPU frequency. More precisely, the power consumption grows superlinearly with the frequency, for a fixed load and number of cores. This is particularly evident in the curves characterizing `Erdos`, which is the most powerful among our servers.

From the previous figures it emerges that the power consumption due to CPU and baseline can be minimized by selecting the right number of active cores and a suitable CPU frequency. Similarly, we can expect that the energy efficiency, defined as number of active cycles per energy unit, can be maximized by tuning the same operational parameters. We graphically represent the impact of operation parameters on power consumption and energy efficiency in Figures 6.2 and 6.4 respectively for `Nemesis` and `Erdos` (results for `Survivor` are similar to the ones shown for `Nemesis` and are omitted).

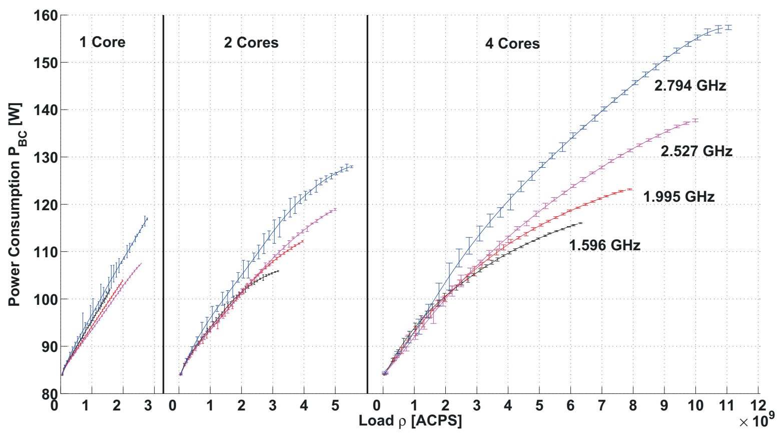
In particular, Figures 6.2(a) and 6.4(a) report all possible fitting curves for the power consumption measurements, plus a curve marking the lowest achievable power consumption at a given load. We name such a curve “minimal power curve” $P_{\min}(\rho)$, and we observe that (i) it only depends on the load ρ, and (ii) it is a piecewise concave function, which makes it suitable to formulate power optimization problems. Finally, to evaluate the energy efficiency of the CPU, we report in Figures 6.2(b) and 6.4(b) the number of active cycles per energy unit obtained from our measurements respectively for `Nemesis` and `Erdos`. We compute the power due to active cycles as the power $P_{BC} - \alpha_0$, i.e., by subtracting the baseline consumption from P_{BC}, and we obtain the efficiency η_C by dividing the load (in active cycles per second) by the power due to active cycles:

$$\eta_C = \frac{\rho}{P_{BC}(\rho) - \alpha_0}. \tag{6.2}$$

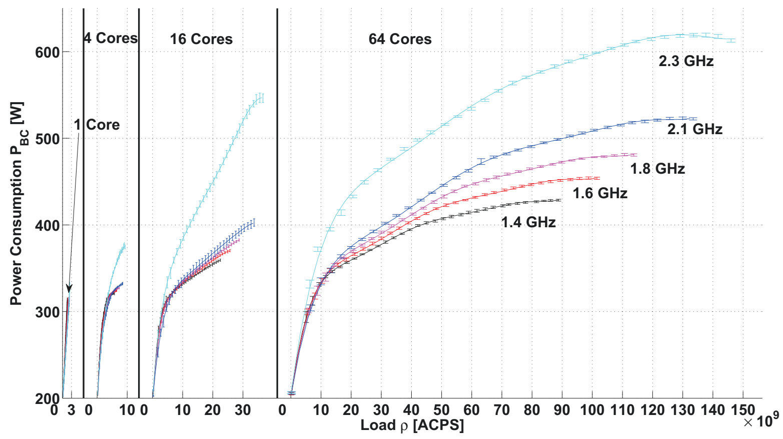
Also in this case we show the curve that maximizes the efficiency at a given load, which we name “Maximal efficiency curve” $\eta_{\max}(\rho)$. Interestingly, we observe that (i) $\eta_{\max}(\rho)$ presents



(a) `Survivor`

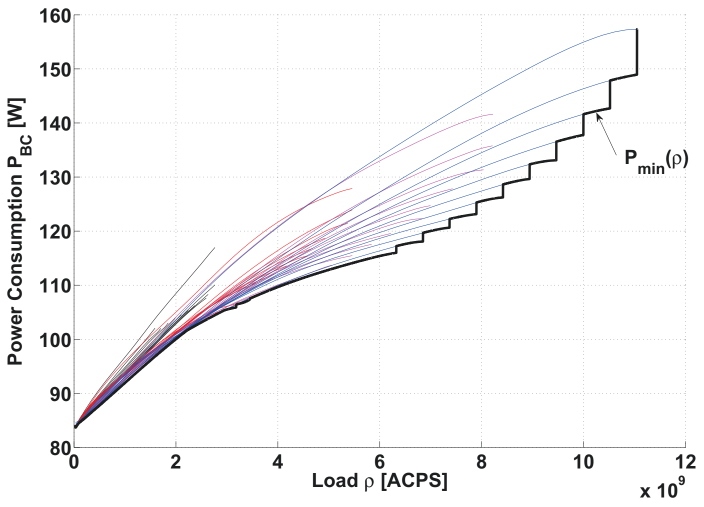


(b) `Nemesis`

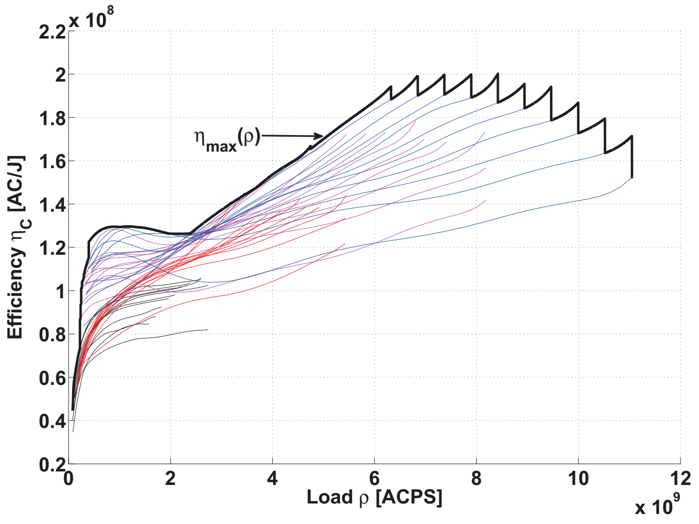


(c) `Erdos`

Figure 6.1: Power consumption of 3 servers (`Survivor`, `Nemesis`, and `Erdos`) for baseline and CPU characterization experiments.

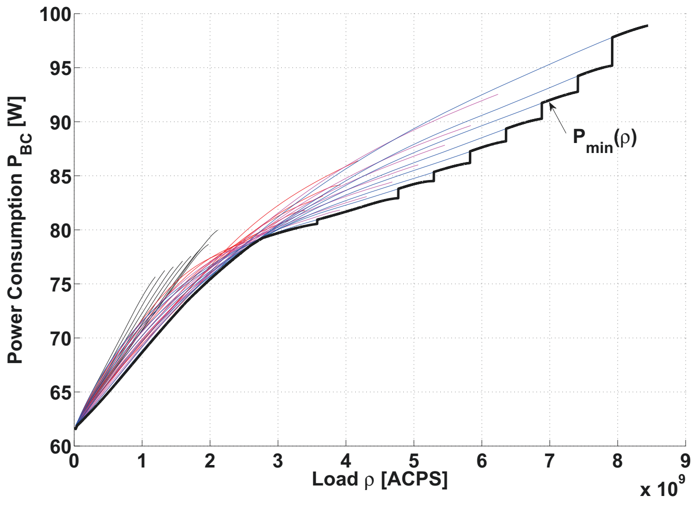


(a) Minimal power.

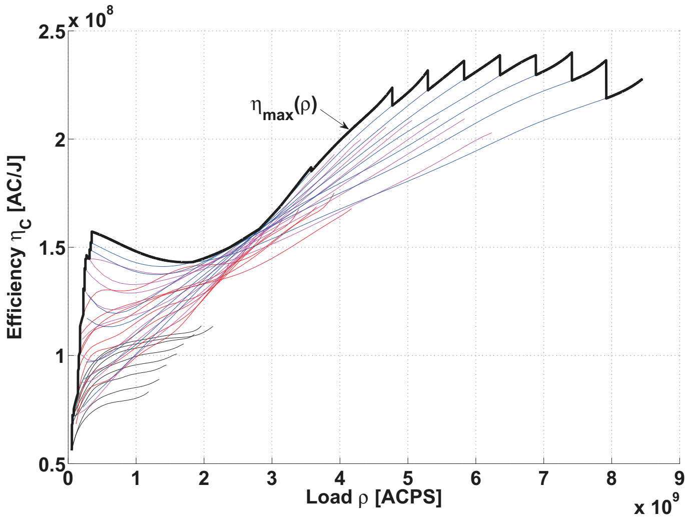


(b) Maximal efficiency.

Figure 6.2: CPU performance bounds of `Nemesis`.

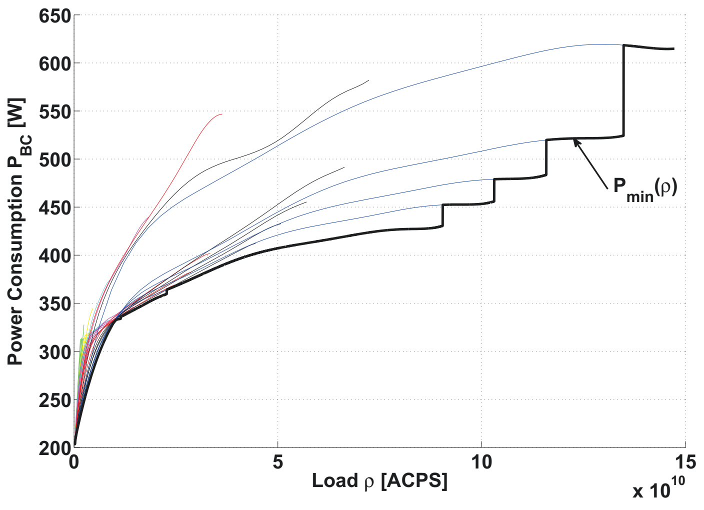


(a) Minimal power.

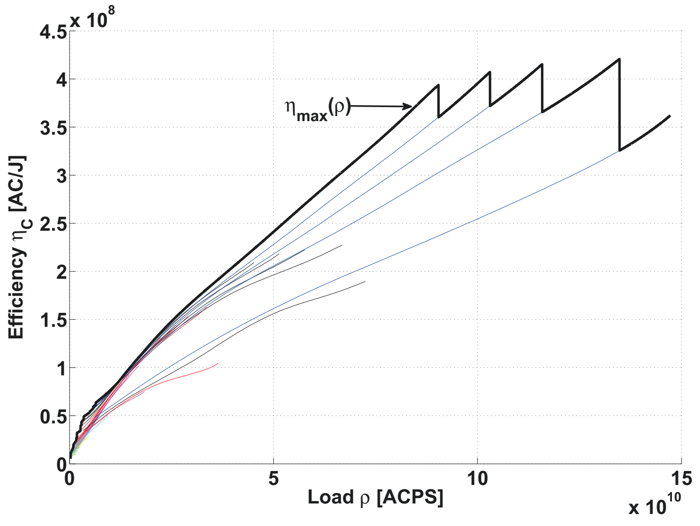


(b) Maximal efficiency.

Figure 6.3: CPU performance bounds of `Survivor`.



(a) Minimal power.



(b) Maximal efficiency.

Figure 6.4: CPU performance bounds of `Erdos`.

multiple local maxima, (ii) for a given configuration of frequency and number of active cores, the efficiency is maximized at the highest achievable load, (iii) all local maxima corresponds to the use of all available active cores, but (iv) the absolute maximum is *not* achieved neither at the highest CPU frequency nor at the lowest.

6.3.3 Disks

We now characterize the power and energy consumption of disk I/O operations. During the experiments, we continuously commit either read or write operations, while keeping the CPU load ρ as low as possible (i.e., we disconnect the network and we do not run other tasks). Still, the power measurements obtained during the disk experiments contain both the power used by the disk and power due to CPU and baseline. Indeed, Figure 6.5 shows, for each experiment, the total measured power P_t, the power P_{BC} computed according to Eq. 6.1 at the load ρ measured during the experiment, and the power due to disk operations, computed as:

$$P_D^x = P_t - P_{BC}(\rho), \quad x \in \{r, w\}, \tag{6.3}$$

where superscripts r and w refer to reading and writing operations, respectively. We test sequentially all the available frequencies for each server (see Table 6.1), and I/O block sizes ranging from 10 *KB* to 100 *MB*. Figure 6.5 shows average and standard deviation of the measures over 10 experiment repetitions for each one of our servers. Indeed, it can be easily seen that `Survivor` and `Nemesis` have similar disks and file systems, while `Erdos` is equipped with SAS disks with RAID. In all cases shown in the figure, the disk power is small but not negligible with respect to the baseline consumption. Furthermore, we can observe that the two servers presented behave differently. Indeed, while the power consumption due to writing is affected both by the block size B for both machines, we observe that both `Nemesis` and `Survivor`' disk writing power P_D^w is not affected by the CPU frequency, while `Erdos`' results show an increase with the frequency. Moreover, the results obtained with `Erdos` are affected by a substantial amount of variability in the measurements, which we believe is due to the caching operations enforced by the RAID mechanism in `Erdos`.

Similarly to what was described for the CPU, we now comment on the energy efficiencies η_D^r and η_D^w of disk reading and writing operations. Figure 6.6 reports efficiency as a function of the I/O block size, and shows one line per each CPU frequency[7]. The efficiency is computed by subtracting the baseline power from the total power, and by measuring the volume V of data read or written in an interval T:

$$\eta_D^x = \frac{V}{P_D^x T}, \quad x \in \{r, w\}. \tag{6.4}$$

We can observe that results are similar for all the servers. Specifically, reading efficiency is almost constant at any frequency and for each block size, while writing is more efficient with large block

[7]For readability, results for `Survivor` are omitted.

sizes. We also observe that the efficiency changes very little with the adopted CPU frequency. Another observation is that the efficiency saturates to a disk-dependent asymptotic value, which is due to the mechanical constraints of the disk (e.g., due to the non-negligible *seek* time, the number of read/write operations per second is limited). In addition, although not visible in the figure due to the log-scale adopted, η_D^w is a concave function of the block size B.

6.3.4 Network

The last server component that we characterize via measurements is the network card. Similarly to the cases described previously, we run experiments in which only the operating system and our test scripts are active. In this case, we run a script to either transmit or receive UDP packets over a gigabit Ethernet connection and count the system active cycles ρ. We measure the total power consumption P_t during the experiment, so that the power due to network activity can be then estimated as follows:

$$P_N^x = P_t - P_{BC}(\rho), \quad x \in \{s, r\}, \tag{6.5}$$

where superscripts s and r refer to the sender and the receiver cases, respectively.

In the experiments, we sequentially test all the available frequencies for each server (see Table 6.1), and fix the packet size and the transmission rate within the achievable set of rates (which depends on the packet size, e.g., < 950 *Mbps* for 1470-B packets). We report results for the network energy consumption in terms of efficiencies η_N^s and η_N^r (volume of data transferred per unit of energy). These efficiencies are computed as follows:

$$\eta_N^x = \frac{R}{P_N^x}, \quad x \in \{s, r\}, \tag{6.6}$$

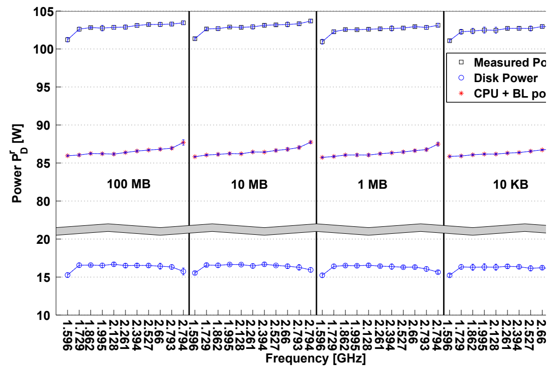
where R is the transmission rate during the experiment.

Figures 6.7, 6.8 and 6.9 show the network efficiencies of `Survivor`, `Nemesis` and `Erdos`, respectively, averaged over 5 samples per transmission rate R.[8] [9] For the sake of readability, the figures only show results for the biggest and smallest packet sizes, i.e., 64-B and 1470-B packets. For `Nemesis` and `Survivor` we report four CPU frequencies: the lowest, the highest, the most efficient (according to Figures 6.2(b) and 6.3(b)) and an intermediate one. For `Erdos` all five available frequencies are shown. The figure also reports the polynomial fitting curves for efficiency, which we found to be at most of second order. Since the efficiency is represented in terms of network activity only, in the fitting we force the zero-order coefficient of the polynomials to be 0. Therefore, we can use the following expression to characterize the network efficiencies of our servers:

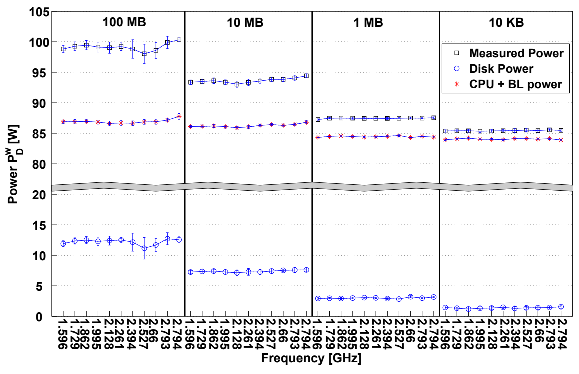
$$\eta_N^x = \beta_1 R + \beta_2 R^2, \quad x \in \{s, r\}, \tag{6.7}$$

[8]Network results are obtained by using a point-to-point Ethernet connection between two controlled servers.

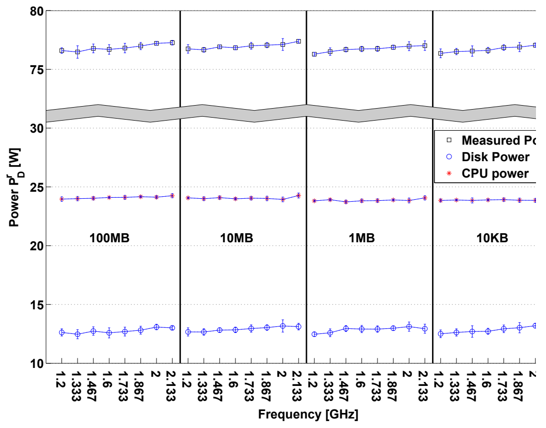
[9]Due to technical and regulation reasons it was only possible to complete the *sender* part for `Erdos`, obtaining only partial results which, because of this partiality, are not published.



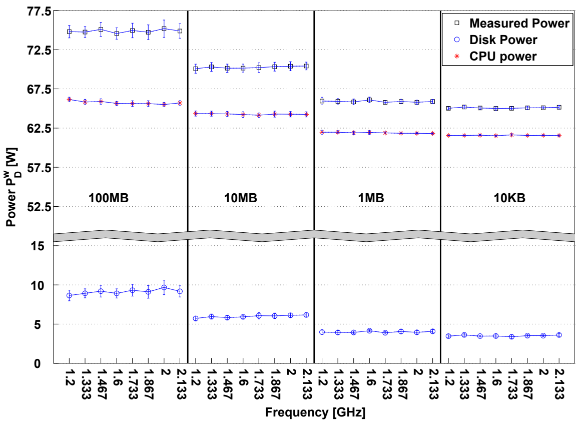
(a) Power consumption during reading (Nemesis



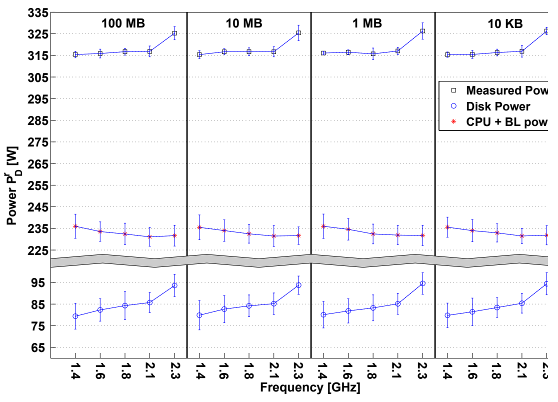
(b) Power consumption during writing (Nemesis).



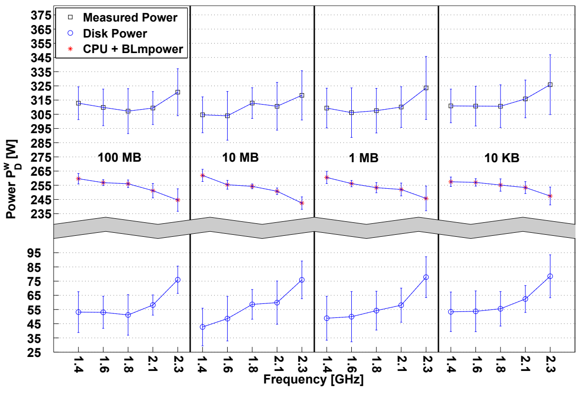
(c) Power consumption during reading (Survivor).



(d) Power consumption during writing (Survivor).



(e) Power consumption during reading (Erdos).



(f) Power consumption during writing (Erdos).

Figure 6.5: Instantaneous power consumption for reading/writing operations. Results are presented for every frequency and for 4 different block sizes for each one of our servers.

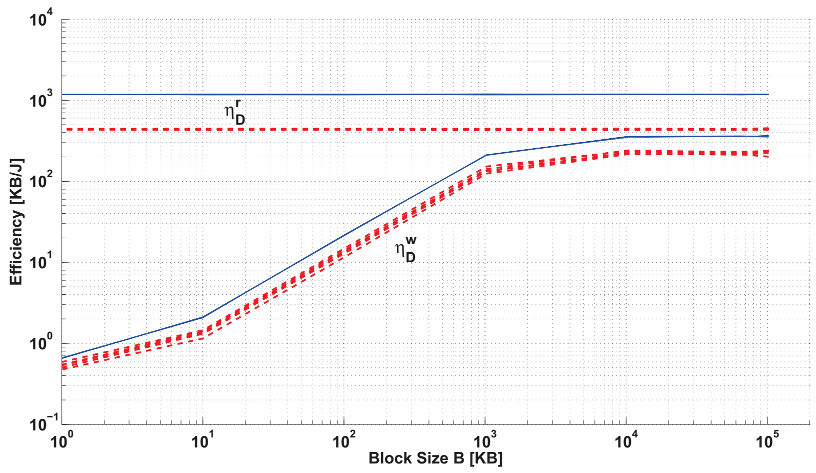


Figure 6.6: Disk reading and writing efficiencies for `Erdos` (red dotted lines) and `Nemesis` (blue solid lines).

where the β_i coefficients are computed by minimizing the least square error of the fitting.

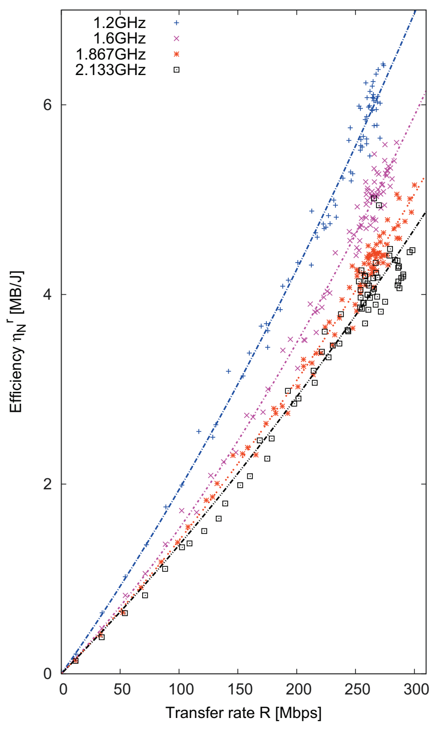
It can observed in Figures 6.7, 6.8 and 6.9 that efficiencies are almost linear or slightly superlinear with the transfer rate, e.g., the receiving efficiency of `Survivor` exhibits an evident quadratic behavior. Indeed, our measurements show that the network power consumption is independent from the throughput, which is a well known result for legacy Ethernet devices. In fact, the NICs of our servers are not equipped with power saving features like, e.g., the recently standardized IEEE 802.3az [92].

In all cases, the efficiency is strongly affected by the selected CPU frequency. Moreover, efficiency is also affected by packet size, although the impact of packet size changes from server to server, e.g., `Survivor` sending efficiency is only slightly affected by it.

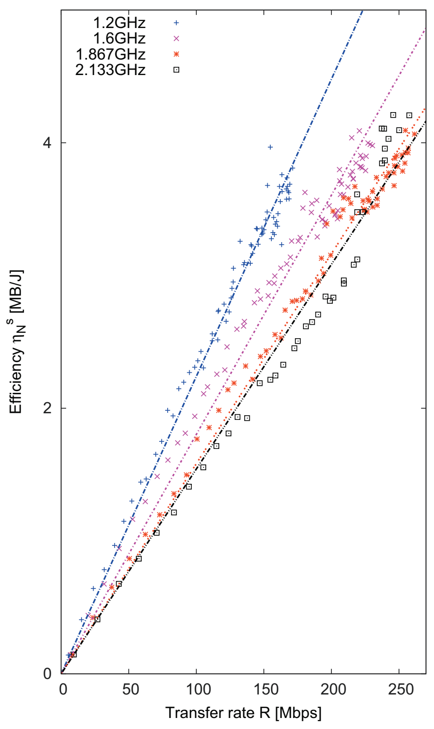
Another observation is that, depending on the packet size and frequency used, sending can be more energy efficient than receiving at a given transmission rate, and using the highest CPU frequency is never the most efficient solution. Note also that the efficiency decreases with the packet size, although this effect is particularly evident at the receiver side, while it only slightly impacts the efficiency of the packet sender. However, network activity also causes non-negligible CPU activity, as shown in Figure 6.10 for a few experiment configurations for all three servers. Overall, the lowest CPU frequency yields the lowest total power consumption during network activity periods.

6.4 Estimating Energy Consumption

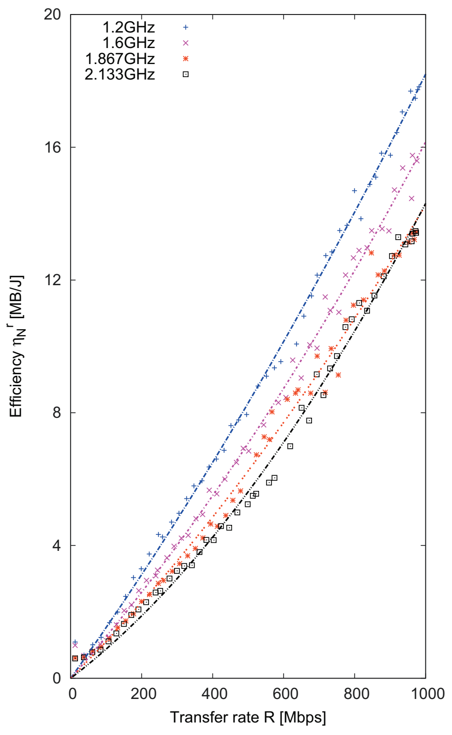
While the results presented in the previous sections are useful to understand the energy consumption pattern of CPU, disk and network, we believe that a much more important use of these results is to estimate the energy consumption of applications. In this section we describe how this can be done from simple data about the application. Moreover, we validate the proposed approach



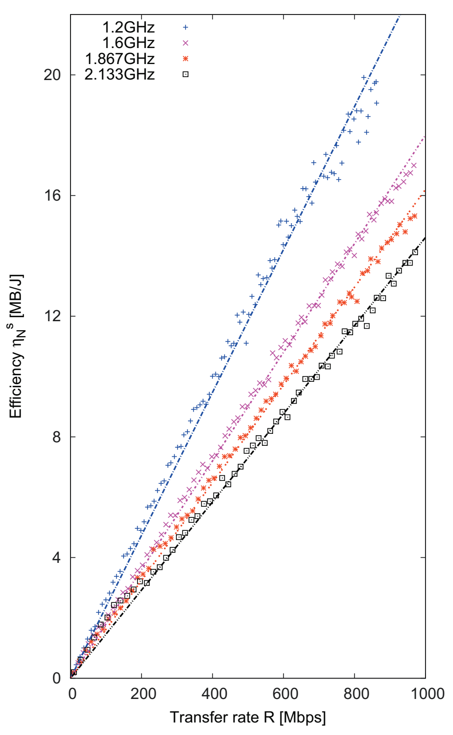
(a) Receiver efficiency in `Survivor` when using 64-B packets.



(b) Sender efficiency in `Survivor` when using 64-B packets.

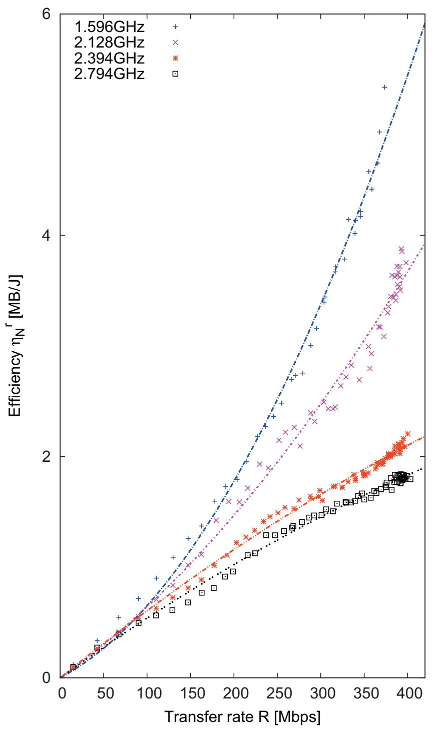


(c) Receiver efficiency in `Survivor` when using 1470-B packets.

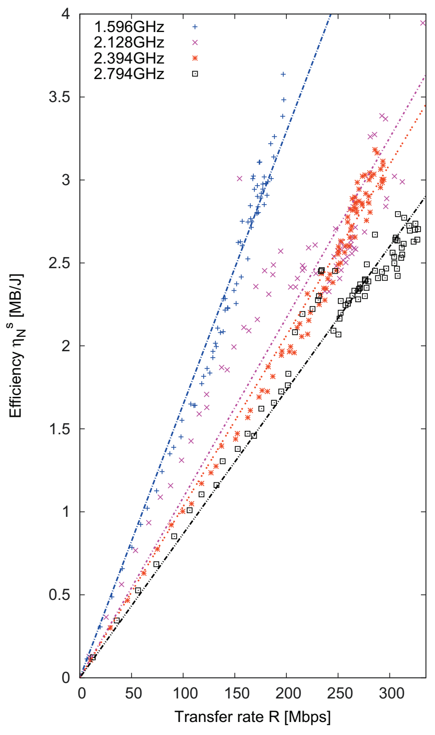


(d) Sender efficiency `Survivor` when using 1470-B packets.

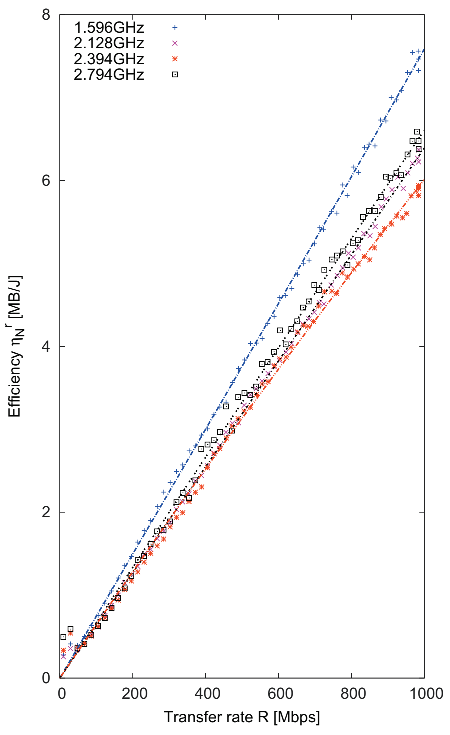
Figure 6.7: Network efficiencies for `Survivor` under different frequencies and 64-B and 1470-B packets.



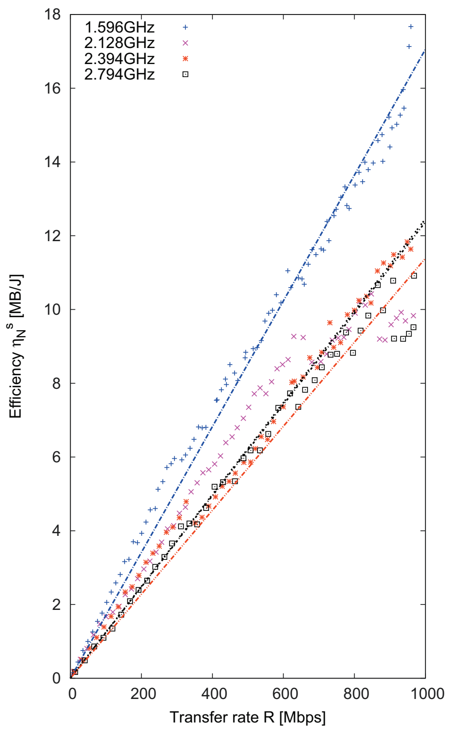
(a) Sender efficiency in `Nemesis` when using 64-B packets.



(b) Sender efficiency in `Nemesis` when using 64-B packets.

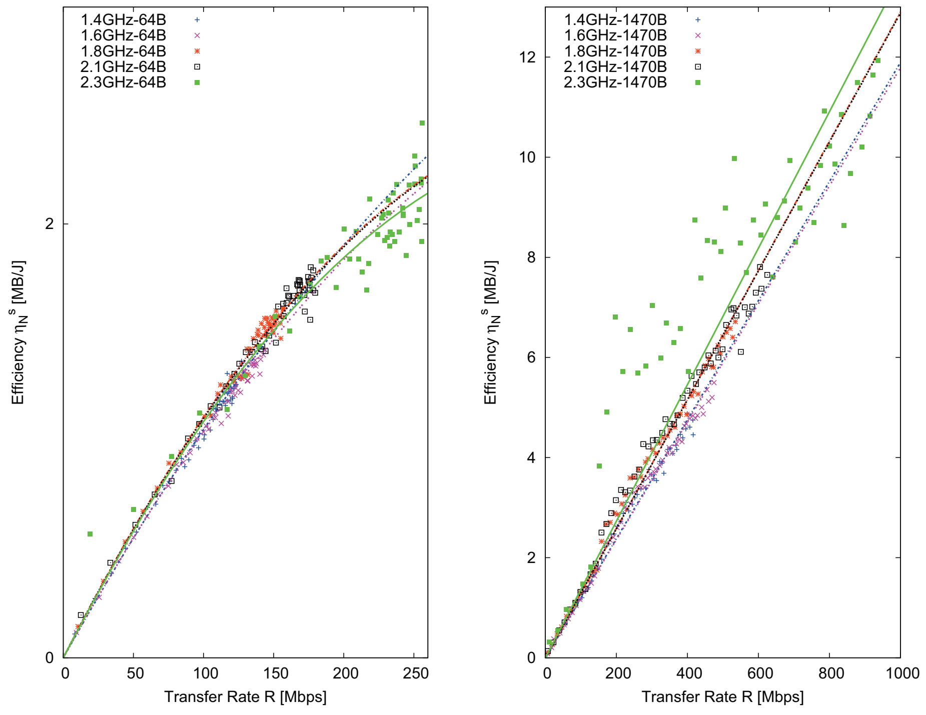


(c) Sender efficiency in `Nemesis` when using 1470-B packets.



(d) Sender efficiency in `Nemesis` when using 1470-B packets.

Figure 6.8: Network efficiencies for `Nemesis` under different frequencies and 64-B and 1470-B packets.



(a) Sender efficiency in `Erdos` when using 64-B packets.

(b) Sender efficiency in `Erdos` when using 1470-B packets.

Figure 6.9: Network efficiencies for `Erdos` under different frequencies and 64-B and 1470-B packets.

by estimating the energy consumed by several *map-reduce Hadoop* computations.

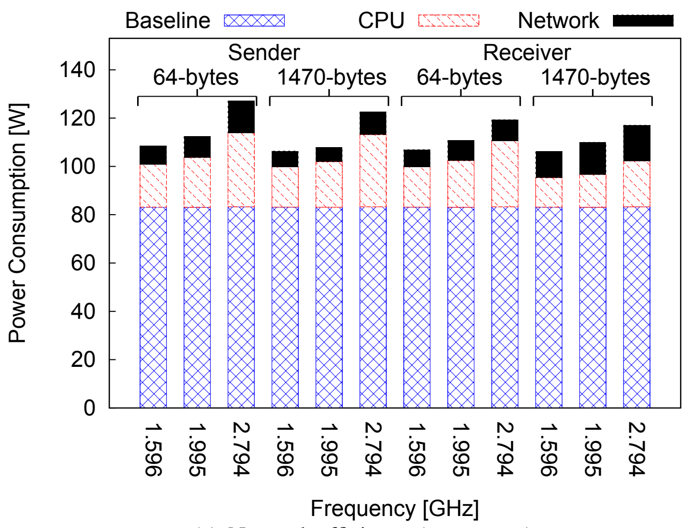
6.4.1 Energy Estimation Hypothesis

The approach we propose to estimate the energy E_{app} consumed by an application lays on the basic assumption that the energy is essentially the sum of the baseline energy E_B (baseline power times application running time), the energy consumed by the CPU E_C, the energy consumed by the disk E_D, and the energy consumed by the network interface E_N:

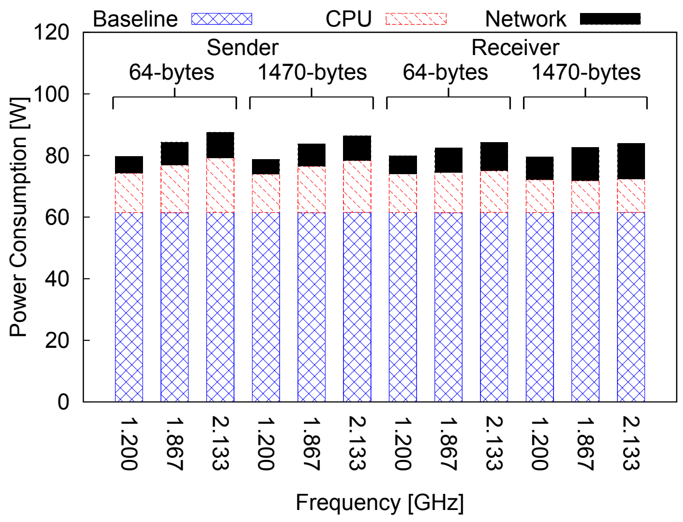
$$E_{app} = E_B + E_C + E_D + E_N. \tag{6.8}$$

Hence, the process of estimating E_{app} is reduced to estimating these four terms. In order to estimate the first two terms, we need to know the total number of active cycles that the application will execute, C_{app}, and the load ρ_{app} (in ACPS) that the execution will incur in the CPU. From this, the total running time T_{app} can be computed as

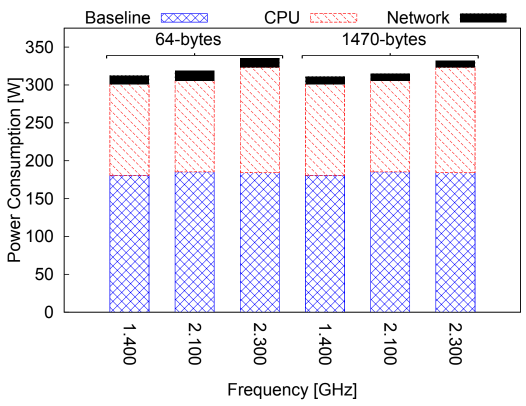
$$T_{app} = C_{app}/\rho_{app}.$$



(a) Network efficiency (`Nemesis`).



(b) Network efficiency (`Survivor`).



(c) Network efficiency (`Erdos`, only sender side).

Figure 6.10: Power utilization with network activity for `Erdos`, `Nemesis` and `Survivor` (64-B experiments were run with a transmission rate $R = 150$ *Mbps*, while $R = 400$ *Mbps* for the experiments with 1470-B packets).

Then, once the number of cores and the frequency that will be used have been defined, it is also possible to estimate the baseline power plus CPU power, P_{BC}, from the fitting curves of Figure 6.1. This allows to estimate the sum of the first two terms of Eq. 6.8 as

$$E_B + E_C = P_{BC} T_{app} = P_{BC} C_{app} / \rho_{app}. \tag{6.9}$$

The energy consumed by the disk is simply the energy consumed while reading and writing, i.e., $E_D = E_D^r + E_D^w$. To estimate these latter values, the block size to be used has to be decided, from which we can obtain an estimate of the efficiency of reading, η_D^r, and writing, η_D^w (see Figure 6.6). These, combined with the total volume of data read and written by the application, denoted as V_D^r and V_D^w respectively, allow to obtain the estimate energy as

$$E_D = \frac{V_D^r}{\eta_D^r} + \frac{V_D^w}{\eta_D^w}. \tag{6.10}$$

Finally, to estimate E_N, the transfer rate R and the packet size S have to be chosen, which combined with the frequency used, yield sending and receiving efficiencies η_N^s and η_N^r (see Figures 6.7, 6.8 and 6.9). Then, if the total volumes of data to be sent and received are V_N^s and V_N^r, respectively,

$$E_N = \frac{V_N^s}{\eta_N^s} + \frac{V_N^r}{\eta_N^r}. \tag{6.11}$$

All is left to do to obtain the estimate E_{app} is to add up the values obtained in Equations 6.9, 6.10, and 6.11.

6.4.2 Applications and Scenarios for Validation

In this subsection we present the applications and scenarios we experimented with in order to validate the model presented in Section 6.4.1. Our goal was to be able to estimate the energy consumed by an application deployed on a data center based on the usage of its different components. For that, we executed two different Hadoop applications, PageRank and WordCount, in three different scenarios: first with an Isolated Server (no network), second with a server connected to the network, and finally with a two-server cloud. For the first two scenarios we used `Nemesis`, whereas, for the cloud case, we used both `Nemesis` and `Survivor`. We describe applications and scenarios in detail below.

Our first application is a **Hadoop Map-Reduce PageRank** based application that follows the approach from Castagna [46]. This application, that we denote PageRank for simplicity, computes several iterations of the pagerank algorithm on an Erdos-Renyi random (directed) graph with 1 million nodes and average degree 5 (In the cluster scenario we used 2 instances of this graph as input in order to use both our nodes during the simulation). The execution of the PageRank application has three phases: preprocessing, map-reduce, and postprocessing. On its side, the map-reduce phase is a sequence of several homogeneous iterations of the PageRank algorithm that runs until a certain threshold is met. For simplicity, we only estimate the energy consumed during the map-reduce phase of the pagerank algorithm, which we force to run 10 times.

Our second application is the **Hadoop Map-Reduce WordCount**. This is a simple program that reads text files and counts how often words occur. For WordCount we use a few hundreds of books as input and estimate the energy consumed for the whole map-reduce process.

As we have mentioned above, these applications are run in 3 different scenarios. In the first scenario, denoted as **Isolated Server**, we run Hadoop in `Nemesis` keeping it disconnected from the network. When we run our applications in this scenario we are basically measuring the impact on the energy consumption of the baseline, CPU and hard disk.

In the second scenario, denoted as **Connected Server**, we run Hadoop in `Nemesis` while it exchanges data on a gigabit LAN. In order to measure the effect of the network on the energy consumption, we evaluate 4 different cases for each application. These cases result from combining 2 different behaviors, depending on whether `Nemesis` acts as a sender or as a receiver of data, with 2 different packet sizes, 64 and 1470 bytes. To do so, we run Iperf, as a server or as a client according to the case, in parallel with Hadoop.

Finally, in the third scenario, denoted as **Cloud**, we set up a two-server Hadoop cluster with `Nemesis` and `Survivor`. In this scenario `Nemesis` is configured as the master node of the cluster and `Survivor` as a slave node. The execution of the applications is shared by both nodes so Hadoop itself exchanges traffic between both servers, and we do not insert additional network traffic in this case. Finally, in order to have a better control of the experiment, we force the reduce tasks to be mandatorily run in `Nemesis`, which also conditions the way the data is exchanged between `Nemesis` and `Survivor`.

Observe that all 3 scenarios are based on Hadoop. This implies that, apart from the map and reduce tasks due to the applications being run, there are some extra processes executed in the servers we are using. The most important processes that we can find in `Nemesis` are *NameNode* (the process that keeps the directory tree of all files in the file system, and tracks where across the cluster the file data is kept), *Secondary NameNode* (that performs periodic checkpoints of the *NameNode*), *DataNode* (the process that is in charge of storing data in the Hadoop File System (HDFS)), *JobTracker* (that receives the jobs and submits MapReduce tasks to the cluster nodes) and *TaskTracker* (a per node process that can accept a determined number of MapReduce tasks). On its side, `Survivor` runs, in the cloud scenario, DataNode and TaskTracker.

6.4.3 Experiments and Observed Results

For the sake of consistency in the results, we ran both applications 10 times per frequency for each one of the considered scenarios and averaged the results.

We start by describing the *Isolated Server* scenario. For each run i we record the total number of active cycles executed C^i_{app}, the time consumed T^i_{app} and the volume of data read (written), $V_D^{r,i}$ ($V_D^{w,i}$). Since we cannot measure the instantaneous CPU load, we assume that the CPU load is the same during the run for a given frequency. Hence, the CPU load can be estimated as

$$\rho^i_{app} = C^i_{app}/T^i_{app}.$$

Then, from ρ^i_{app} we obtain the estimate of the instantaneous power P^i_{BC} using the fitting curves as described in Section 6.3. Finally, using Eq. 6.9 we compute the estimate $E^i_B + E^i_C$. In order to estimate the energy consumed by the disk operations, we use the fact that Hadoop uses a block size of 64 MB. This allows us to estimate the reading (writing) efficiencies, $\eta^{r,i}_D$ ($\eta^{w,i}_D$) that we compute, in Joules per byte. Combining these values with the measured volume of data read and written ($V^{r,i}_D$ and $V^{w,i}_D$), as described in Eq. 6.10, we obtain E^i_D.

The total estimated energy of the application in run i, E^i_{app}, is obtained by summing up the energy of the different components used in run i, as stated in Eq. 6.8 (remember that, in the *Isolated Server* the network is not used). Then, by summing the values of the ten runs of an experiment, we obtain the total estimated energy as

$$E_{app} = \sum_{i=1}^{10} E^i_{app}.$$

The (approximated) total *real* energy $\hat{E}^i_{app}$ consumed in run i is computed by the average value of the power samples which we registered with the power analyzer during the run, and we multiply it with the run time T_{app}. Finally, the total energy consumed by the experiment is obtained as

$$\hat{E}_{app} = \sum_{i=1}^{10} \hat{E}^i_{app}.$$

The estimation error for each experiment is then computed as $\hat{E}_{app} - E_{app}$.

We show the results obtained for the *Isolated Server* scenario with the minimum, the maximum, and the most efficient[10] frequencies (the results for the remaining frequencies are similar) in Figure 6.11. The figure shows the results for both PageRank and WordCount. As can be seen, the error is relatively small, except for the case when we run WordCount at the maximum frequency. Errors are of 4%, 4%, 7%, 5%, 7% and 10% respectively, following the same order as in Figure 6.11.

We move now to the *Connected Server* scenario. As we described in the previous section, this scenario is studied in 4 different cases depending on whether `Nemesis` acts as sender or receiver and whether the size of the packets is of 64 or 1470 bytes. Of course, another relevant parameter is the rate at which these packets are sent. The rates used are 150 and 400 Mbps when using packets of 64 or 1470 bytes, respectively.

The total energy consumed in these cases is computed in the same way as we did for the *Isolated Server* scenario but adding the contribution of the network. In order to estimate the net-

[10]According to the results shown in Section 6.3.

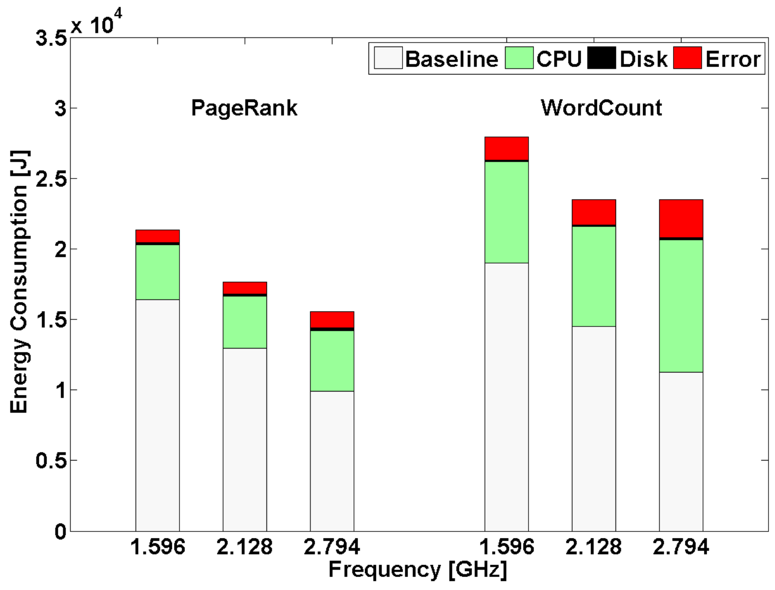


Figure 6.11: Energy consumption of `Nemesis` in the Isolated Server scenario.

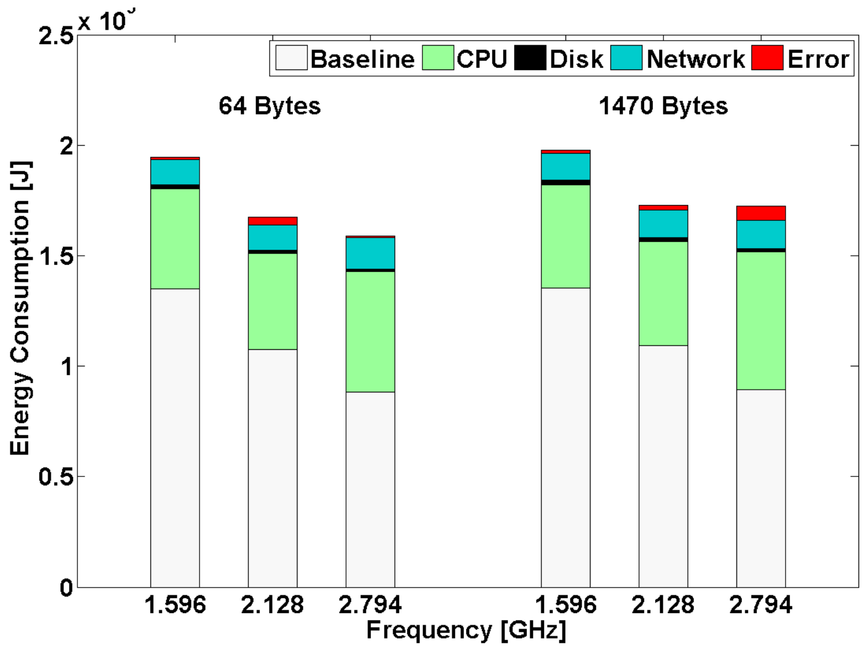
Table 6.2: Error measured in the different cases of the *Connected Server* scenario.

Packet Size	Freq	Cases			
		PR - Send	PR - Rec	WC - Send	WC - Rec
64-B	1.596	0.5%	6.0%	2.9%	2.7%
	2.128	2.0%	4.7%	6.4%	1.5%
	2.794	0.5%	2.9%	4.0%	2.9%
1470-B	1.596	0.7%	6.9%	1.6%	6.5%
	2.128	1.1%	6.5%	5.8%	5.8%
	2.794	3.8%	0.3%	0.9%	1.5%

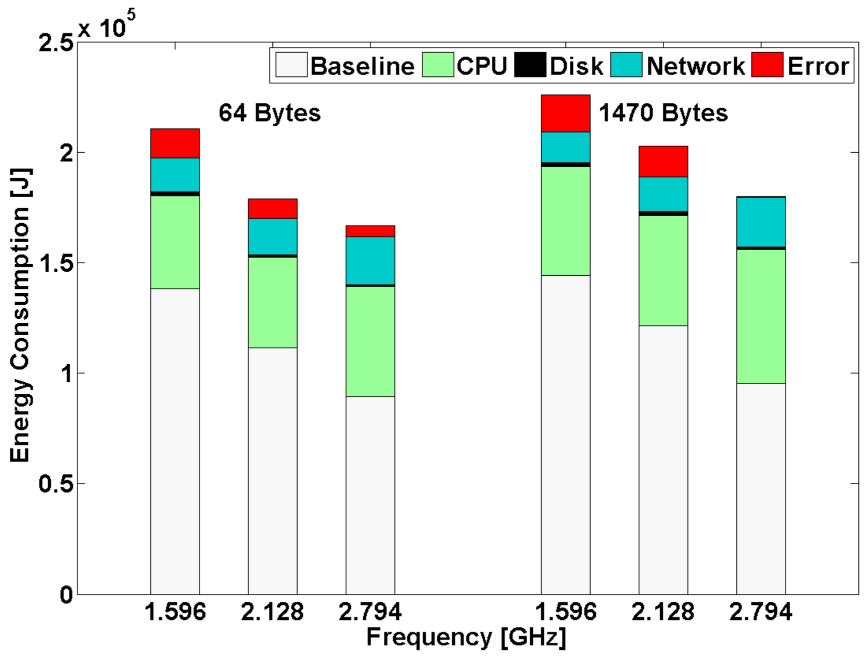
work consumption in one run with `Nemesis` sending traffic (resp., receiving traffic), the sending efficiency η_N^s, (resp., receiving efficiency η_N^r) is obtained from the transfer rate R, the frequency and packet size used (see Figures 6.8). The amount of data sent (resp., received) can be obtained from the server itself by consulting the OS registers[11]. Therefore, the energy of the network for an run i, E_N^i, is obtained using Eq. 6.11. Then, including E_N^i for each run in the computation of E_{app}^i we can obtain the total energy consumed by the application. Following the same steps as in the previous scenario, we get the results shown in Figure 6.12 and 6.13. The error measured is again relatively smaller for PageRank than for WordCount. The error measured for each of the cases can be found in Table 6.2.

We finally analyze the *Cloud* scenario. In this scenario we set up a cluster with two servers, `Nemesis` and `Survivor`, and run the 2 aforementioned Hadoop applications in it. This scenario may seem relatively similar to the *Connected Server* scenario, but it has is a major difference. While in the previous scenario we were the ones controlling the network traffic, here the traffic is controlled by Hadoop. Specifically, we know that, in this scenario, there are two

[11]We can read the registers rx_bytes, rx_packets, tx_bytes or tx_packets from `/sys/class/net/eth0/statistics`.

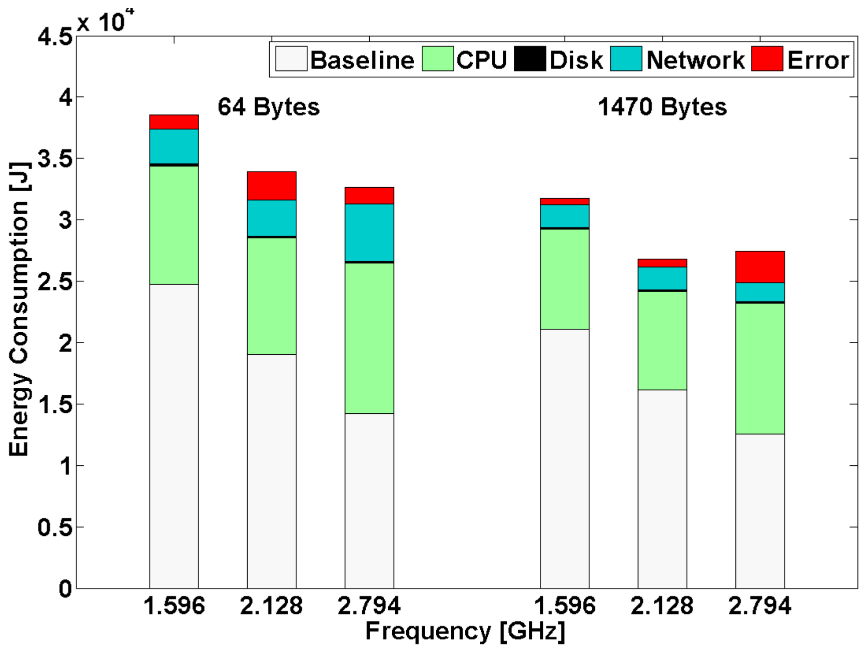


(a) PageRank, sender side.

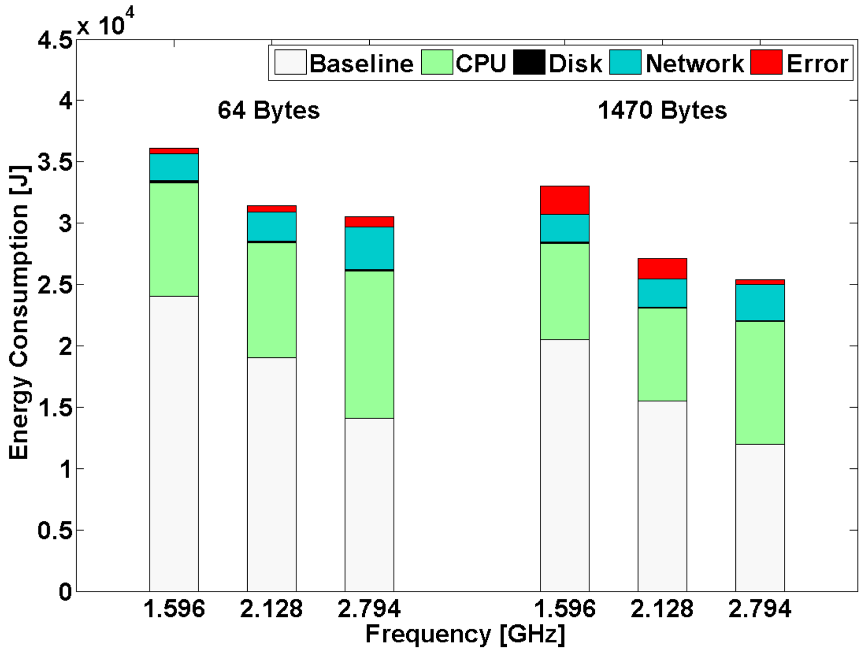


(b) PageRank, receiver side.

Figure 6.12: Energy consumption of Nemesis running PageRank in the Connected Server scenario, with either small or big packets.



(a) WordCount, sender side.



(b) WordCount, receiver side.

Figure 6.13: Energy consumption of `Nemesis` running WordCount in the Connected Server scenario, with either small or big packets.

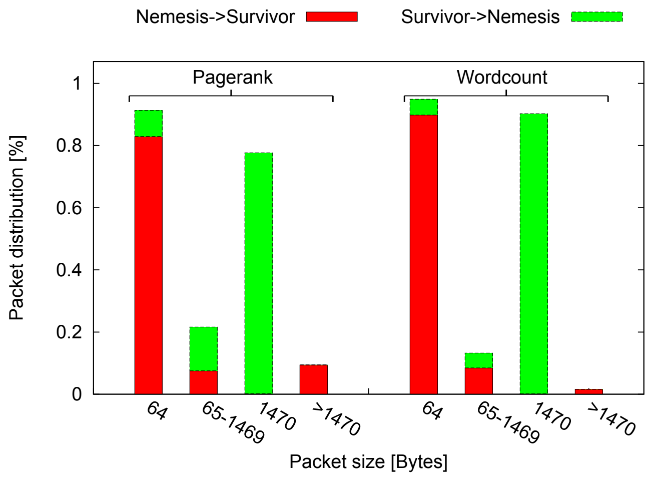
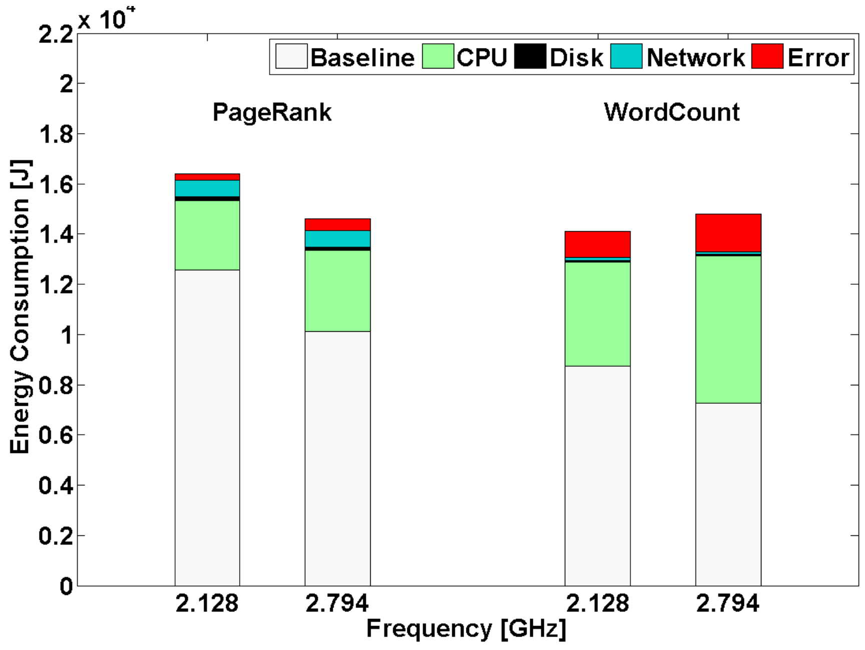


Figure 6.14: Distribution of the sizes of the packets exchanged between `Nemesis` and `Survivor` for both PageRank and WordCount in the Cloud scenario.

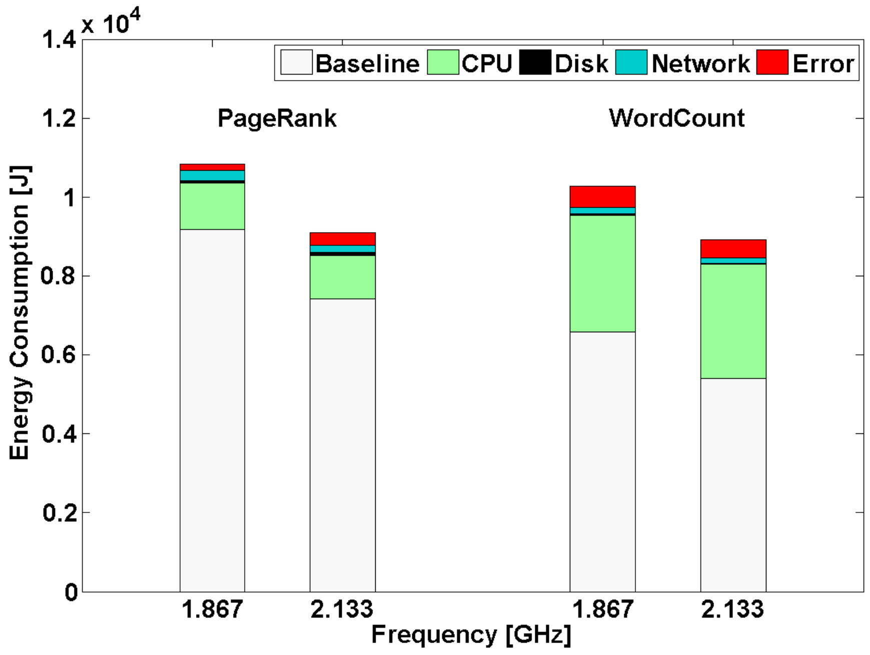
main sources of traffic: requesting input data when it is not present in a server, and sending the mapper tasks outputs to the reducer tasks. The only condition we impose in the server to have some control over the traffic is related to this later aspect, we force the reducers to be always in `Nemesis`.

Although we are able to retrieve the total amount of data received or sent by each server, we know neither the size of the packets used nor the rate. Therefore, we can compute neither the sending efficiency η_N^s nor the receiving efficiency η_N^r. In order to be able to compute both the sending and receiving efficiencies we analyze the traffic exchanged by both servers for each one of the applications. Figure 6.14 shows the amount of packets of each size that were exchanged by both servers (and the direction of the exchange) for both applications. The results show the vast majority of packets are either small (64 bytes) or big (1470 bytes). Moreover, it shows that most of the packets sent from `Nemesis` to `Survivor` are small packets for both applications, while big packets are sent in the opposite direction.

Given these results, we approximate the energy consumed by the network assuming that all the packets exchanged are of the same size and that the rate is the maximum achievable rate for each packet size according to the results from Section 6.3. For instance, we consider roughly 30 Mbps when `Survivor` receives 64-Byte packets and roughly 970 Mbps if it sends 1470-Byte packets. These assumptions allow us to compute now η_N^s and η_N^r. The remaining parameters are computed as for the other scenarios, so to determine $\hat{E}_{app}$ and E_{app}. The results are shown in Figure 6.15. As in the previous scenarios, errors are relatively low. In particular, the error in `Nemesis` when running PageRank is 3.1% and 1.4% for 2.128 GHz and 2.794 GHz, respectively, and of a 9.7% and a 6.5% for 2.128 GHz and 2.794 GHz when running WordCount. On the other hand, the measured errors for `Survivor` are 3.3% and 3.6% for 1.867 GHz and 2.133 GHz when running PageRank and 5.1% and 5.2%, respectively, when running WordCount.



(a) `Nemesis`



(b) `Survivor`

Figure 6.15: Energy consumption of `Nemesis` and `Survivor` in the Cloud scenario.

6.5 Discussion

We discuss now some of the implications of our results. We start with consolidation as a technique for energy saving. It has been often assumed that the best way of saving energy is by using the highest frequency available and applying consolidation (which is to fill servers as much as possible). This reduces the total number of servers being used, allowing to switch off the rest. This assumption has led to proposing bin-packing based solutions [33, 124, 164]. However, the results presented in Figures 6.2(b), 6.3(b) and 6.4(b) show that the highest frequency is not always the most efficient one, and this has been found to be true for two different architectures (Intel and AMD). This implies that, by running servers at the optimal amount of load, and the right frequency, a considerable amount of energy could be saved.

A second relevant aspect is the baseline consumption of servers. The results presented for all 3 servers show that their baselines are within a 30-50% of the maximum consumption. Then, it is obvious that more effort has to be done for reducing baseline consumption. For instance, a solution could consist in switching off cores in real time, not just disabling them, or in introducing very fast transitions between active and lower energy states, i.e., to achieve real *suspension* in idle state.

There is another relevant issue related to the CPU load associated to disk and network activity. It can be observed in Figure 6.5 that disks do not incur much CPU overhead. In fact, the power used by the CPU plus baseline does not change much across the experiments. Instead, the energy consumed by the CPU due to network operations is even larger than the energy consumed by the NIC (see Figure 6.10). Some works [74] have already pointed out that the way packets are handled by the protocol stack is not energy efficient. Our results reinforce this feeling and point out that building a more efficient protocol stack would certainly reduce the amount of energy consumed due to the network.

Finally, it is worth to mention that in this work we have assumed that the power utilization of the RAM memory is included in the *baseline*. The characterization experiments have been run in such a way that there were few memory accesses, so its power utilization did not affect our measurements. However, RAM memory became an uncontrolled source of power utilization in Section 6.4.3 when we validated our proposed model. In fact, all the Hadoop processes that run in the servers consume significant RAM memory. This impacts more significantly the memory used by the cluster's master node, since it runs internal Hadoop processes (such as the NameNode or the JobTracker) whose memory requirement increases with the number of mappers and reducers. This cost is, therefore, paid only in `Nemesis`, the master node of our cluster, and not in `Survivor`, which explains the different accuracy of the model for the two servers. This error is particularly evident when when WordCount is run, due to the fact that the required number of mappers for WordCount is larger than for PageRank and, therefore, the RAM required in `Nemesis` increases and so does the uncontrolled energy consumption.

6.6 Conclusions

In this chapter we have reported our measurement-based characterization of energy and power consumption in a server. We have exhaustively measured the power consumed by CPU, disk, and NIC under different configurations, identifying the optimal operational levels, which usually do not correspond to the static system configurations commonly adopted. We found that, besides the *baseline component*, which does not changes significantly with the operational parameters, the CPU has the largest impact on energy consumption among all the three components. We observe that CPU consumption is neither linear nor concave with the load, i.e., the systems are not *energy proportional*. Disk I/O is the second larger contributor to power consumption, although performance changes sensibly with the I/O block size used by the applications. Finally, the NIC activity is responsible for a small but not negligible fraction of power consumption, which scales almost linearly with the network transmission rate. In general, most of the energy/power performance figures do not scale linearly with the utilization, in contrast to what is commonly assumed in the literature. We have then shown how to predict and optimize the energy consumed by an application via a concrete example using 2 different Hadoop applications, PageRank and WordCount, in three different scenarios. First We ran both applications without network activity, next with bulky network activity, and finally in a two-server cluster. Our model achieves very accurate energy estimates, within 4.1% from the measured total power consumption on average and never worse than a 10%.

Chapter 7

Efficient Assignment of Virtual Machines to Physical Machines

7.1 Overview

Having studied how nowadays servers use power, we now show a way in which this knowledge can be useful. In this chapter we will apply the concept of the optimal operational point of a server to the Virtual Machine Assignment problem (VMA), which basically consists in deciding in which server we want to place a new task arriving to a system. Having an optimal operational point conditions drastically the way we must load a server in order to optimize the energy which is being consumed.

Having this in mind, we study, in particular, the hardness and online competitiveness of the four versions of the VMA problem that we described in Chapter 2. This 4 versions of VMA differed in whether we considered PMs with bounded or unbounded capacity C or a limited or unlimited number of PMs m in the system. We denoted these versions as $(\cdot, \cdot)$-VMA, $(C, \cdot)$-VMA, $(\cdot, m)$-VMA and (C, m)-VMA[1].

We start by showing various lower and upper bounds on the offline approximation of VMA. The first fact we observe is that there is a hard decision version of (C, m)-VMA: Is there a feasible partition π of the set D of VMs? By reduction from the 3-Partition problem, it can be shown that this decision problem is strongly NP-complete. We then show that the $(\cdot, \cdot)$-VMA, $(C, \cdot)$-VMA, and $(\cdot, m)$-VMA problems are NP-hard in the strong sense, even if α is constant. This result implies that VMA problems do not have a fully polynomial time approximation scheme (FPTAS), even if α is constant. Nevertheless, using previous results derived for more general objective functions, we notice that $(\cdot, \cdot)$-VMA and $(\cdot, m)$-VMA have a polynomial time approximation scheme (PTAS), while the $(C, \cdot)$-VMA problem can not be approximated beyond a ratio of $\frac{3}{2} \cdot \frac{\alpha - 1 + (\frac{2}{3})^\alpha}{\alpha}$ (unless P $=$ NP). On the positive side, we show how to use an existing Asymptotic PTAS [69] to obtain algorithms that approximate the optimal solution of $(C, \cdot)$-VMA. All our

[1]The central dots $(\cdot)$ imply unboundedness.

offline results as well as the online ones can be seen in Table 7.1.

Then we move on to online VMA algorithms. We show various upper and lower bounds on the competitive ratio of the four versions of the problem. Observe that the results are often different depending on whether x^* is smaller than C or not. In fact, when $x^* < C$, there is a lower bound of $\frac{(3/2)2^\alpha-1}{2^\alpha-1}$ that applies to all versions of the problem. Rather than attempting to obtain tight bounds for particular instances of the parameters of the problem (C, m, α, b) we focus on obtaining *general bounds*, whose parameters can be instantiated for the specific application. The bounds obtained show interesting trade-offs between the PM capacity and the fixed cost of adding a new PM to the system. For clarity, we will consider $\mu = 1$ throughout the whole chapter. All the results presented apply for other values of μ.

The resulting bounds are shown in Table 7.1. As can be observed, the resulting upper and lower bounds are not very far in general. To give some intuition on the tightness of these bounds, we instantiate them for a realistic value of $\alpha = 3$, and normalized values of $b = 2$ and $C \in \{1, 2\}$. These values for α are obtained from the servers we studied in Chapter 6, in particular from the ones denoted as `Erdos` and `Nemesis`. In them the values for α are close to 1.5 and 3 and x^* values of $0.76C$ and $0.9C$ respectively (x^* denotes the load that minimizes the ratio power consumption against load.). The bounds based on these realistic values are shown in Table 7.2.

RoadMap The rest of the chapter is organized as follows. Section 7.2 includes some preliminary results that will be used throughout the chapter. The offline and online analyses are included in Section 7.3 and 7.4 respectively. In Section 7.5 we compare different state of the art allocation policies and compare them with the algorithms proposed in Section 7.4. Section 7.6 discusses some practical issues and provides some useful insights regarding real implementation. Section 7.7 concludes the Chapter.

7.2 Preliminaries

The following claims will be used in the analysis. We call ***power rate*** the power consumed per unit of load in a PM. Let x be the load of a PM. Then, its power rate is computed as $f(x)/x$. The load at which the power rate is minimized, denoted x^*, is the ***optimal load***, and the corresponding rate is the ***optimal power rate*** $\varphi^* = f(x^*)/x^*$. Using calculus we get the following observation.

Observation 2. *The optimal load is* $x^* = (b/(\alpha-1))^{1/\alpha}$. *Additionaly, for any* $x \neq x^*$, $f(x)/x > \varphi^*$.

The following lemmas will be used in the analysis.

Lemma 13. *Consider two solutions* $\pi = \{A_1, \ldots, A_m\}$ *and* $\pi' = \{A'_1, \ldots, A'_m\}$ *of an instance of the VMA problem, such that for some* $x, y \in [1, m]$ *it holds that*

- $A_x \neq \emptyset$ *and* $A_y \neq \emptyset$*;*

VMA subprob.	$x^* < C$	$x^* \geq C$
$(C, \cdot)$ offline	$\rho \geq \frac{3}{2}\frac{\alpha-1+(2/3)^\alpha}{\alpha}$	$\rho \geq \frac{3}{2}\frac{\alpha-1+(2/3)^\alpha}{\alpha}$
	$\rho < \frac{\overline{m}}{m^*}\left(1+\epsilon+\frac{1}{\alpha-1}+\frac{1}{\overline{m}}\right)$	$\rho < 1+\epsilon+\frac{C^\alpha}{b}+\frac{1}{\overline{m}}$
$(C, \cdot)$ online	$\rho \geq \frac{(3/2)2^\alpha-1}{2^\alpha-1}$	$\rho \geq \frac{C^\alpha+2b}{b+\max\{C^\alpha, 2(C/2)^\alpha+b\}}$
	$\rho = 1$ if $D_s = \emptyset$, else $\rho \leq \left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right)\left(2+\frac{x^*}{\ell(D_s)}\right)$	$\rho \leq \frac{2b}{C}\left(1+\frac{1}{(\alpha-1)2^\alpha}\right)\left(2+\frac{C}{\ell(D)}\right)$
(C, m) online	$\rho \geq \frac{(3/2)2^\alpha-1}{2^\alpha-1}$	$\rho \geq \frac{C^\alpha+2b}{b+\max\{C^\alpha, 2(C/2)^\alpha+b\}}$
$(\cdot, \cdot)$ online	$\rho \geq \frac{(3/2)2^\alpha-1}{2^\alpha-1}$	not applicable
	$\rho = 1$ if $D_s = \emptyset$, else $\rho \leq \left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right)\left(2+\frac{x^*}{\ell(D_s)}\right)$	
$(\cdot, m)$ online	$\rho \geq \max\{\frac{(3/2)2^\alpha-1}{2^\alpha-1}, \frac{3^\alpha}{2^{\alpha+2}+\epsilon}\}$	not applicable
	$\rho \leq O(\alpha)^\alpha$ In [82]	
$(\cdot, 2)$ online	$\rho \geq \max\{\frac{3^\alpha}{2^{\alpha+1}}, \frac{(3/2)2^\alpha-1}{2^\alpha-1}, \frac{3^\alpha}{2^{\alpha+2}+\epsilon}\}$	not applicable
	$\rho = 1$ if $\ell(D) \leq \sqrt[\alpha]{b/(2^\alpha-2)}$, else $\rho \leq \max\{2, \left(\frac{3}{2}\right)^{\alpha-1}\}$	

Table 7.1: Summary of bounds on the approximation/competitive ratio ρ. All lower bounds are existential. The number of PMs in an optimal $(C, \cdot)$-VMA solution is denoted as m^*. The number of PMs in an optimal Bin Packing solution is denoted as $\overline{m}$. The load that minimizes the ratio power consumption against load is denoted as x^*. The subset of VMs with load smaller than x^* is denoted as D_s.

- *$A'_x = A_x \cup A_y$, $A'_y = \emptyset$, and $A_i = A'_i$, for all $i \neq x$ and $i \neq y$; and*
- *$\ell(A_x) + \ell(A_y) \leq \min\{x^*, C\}$.*

Then, $P(\pi') < P(\pi)$.

Proof: Let $\ell(A_i) = x$ and $\ell(A_j) = y$. First we notice that π' is feasible because $x + y \leq C$. Now, using that $x + y \leq x^*$, we have

$$b = (x^*)^\alpha(\alpha-1) \geq (x+y)^\alpha(\alpha-1) > (x+y)^\alpha \geq (x+y)^\alpha - (x^\alpha + y^\alpha)$$

where the second inequality comes from the fact that $\alpha > 1$. The above inequality is equivalent to

$$2b + x^\alpha + y^\alpha > b + (x+y)^\alpha,$$

VMA subprob.	$x^* < C$	$x^* \geq C$
$(C, \cdot)$ offline	$\rho \geq \frac{11}{9}$	$\rho \geq \frac{11}{9}$
	$\rho < \frac{\overline{m}}{m^*}\left(\frac{3}{2} + \epsilon + \frac{1}{\overline{m}}\right)$	$\rho < \frac{3}{2} + \epsilon + \frac{1}{\overline{m}}$
$(C, \cdot)$ online	$\rho \geq \frac{11}{7}$	$\rho \geq \frac{20}{17}$
	$\rho \leq \frac{17}{12}\left(1 + \frac{1}{2\ell(D_s)}\right)$	$\rho \leq \frac{17}{2}\left(1 + \frac{1}{2\ell(D)}\right)$
(C, m) online	$\rho \geq \frac{11}{7}$	$\rho \geq \frac{20}{17}$
$(\cdot, \cdot)$ online	$\rho \geq \frac{11}{7}$	not applicable
	$\rho \leq \frac{17}{12}\left(1 + \frac{1}{2\ell(D_s)}\right)$	
$(\cdot, m)$ online	$\rho \geq \frac{11}{7}$	not applicable
$(\cdot, 2)$ online	$\rho \geq \frac{11}{7}$	not applicable
	$\rho \leq \frac{9}{4}$	

Table 7.2: Summary of bounds on the approximation/competitive ratio ρ for $\alpha = 3$, $b = 2$, and $C = 2$ on the left and $C = 1$ on the right.. All lower bounds are existential. The number of PMs in an optimal $(C, \cdot)$-VMA solution is denoted as m^*. The number of PMs in an optimal Bin Packing solution is denoted as $\overline{m}$. The load that minimizes the ratio power consumption against load is denoted as x^*. The subset of VMs with load smaller than x^* is denoted as D_s.

which implies the lemma. ■

From this lemma, it follows that the global power consumption can be reduced by having 2 VMs together in the same PM, when its aggregated load is smaller than $\min\{x^*, C\}$, instead of moving one VM to an unused PM. When we keep VMs together in a given partition we say that we are *using* Lemma 13.

Lemma 14. *Consider two solutions $\pi = \{A_1, \ldots, A_m\}$ and $\pi' = \{A'_1, \ldots, A'_m\}$ of an instance of the VMA problem, such that for some $x, y \in [1, m]$ it holds that*

- *$A_x \cup A_y = A'_x \cup A'_y$, while $A_i = A'_i$, for all $x \neq i \neq y$;*
- *none of A_x, A_y, A'_x, and A'_y is empty; and*
- *$|\ell(A_x) - \ell(A_y)| < |\ell(A'_x) - \ell(A'_y)|$.*

Then, $P(\pi) < P(\pi')$.

Proof: From the definition of $P(\cdot)$, to prove the claim is it enough to prove that

$$\ell(A_x)^\alpha + \ell(A_y)^\alpha < \ell(A'_x)^\alpha + \ell(A'_y)^\alpha.$$

Let us assume wlog that $\ell(A_x) \leq \ell(A_y)$ and $\ell(A'_x) \leq \ell(A'_y)$. Let us denote

$$L = \ell(A_x) + \ell(A_y) = \ell(A'_x) + \ell(A'_y),$$

and assume that $\ell(A_x) = \delta_1 L$ and $\ell(A'_x) = \delta_2 L$. Note that $\delta_2 < \delta_1 \leq 1/2$. Then, the claim to be proven becomes

$$(\delta_1 L)^\alpha + ((1-\delta_1)L)^\alpha < (\delta_2 L)^\alpha + ((1-\delta_2)L)^\alpha$$
$$\delta_1^\alpha + (1-\delta_1)^\alpha < \delta_2^\alpha + (1-\delta_2)^\alpha$$

Which holds because the function $f(x) = x^\alpha + (1-x)^\alpha$ is decreasing in the interval $(0, 1/2)$. ■

This lemma carries the intuition that balancing the load among the used PMs as much as possible reduces the power consumption.

Corollary 14. *Consider a solution $\pi = \{A_1, \ldots, A_m\}$ of an instance of the VMA problem with total load $\ell(D)$, such that exactly k of the A_x sets, $x \in [1, m]$, are non-empty (hence it uses k PMs). Then, the power consumption is lower bounded by the power of the (maybe unfeasible) solution that balances the load evenly, i.e.,*

$$P(\pi) \geq kb + k(\ell(D)/k)^\alpha.$$

7.3 Offline Analysis

7.3.1 NP-hardness

As was mentioned, it can be shown that deciding whether there is a feasible solution for an instance of the (C, m)-VMA problem is NP-complete or not, by a direct reduction from the 3-Partition problem. However, this result does not apply directly to the $(C, \cdot)$-VMA, $(\cdot, m)$-VMA, and $(\cdot, \cdot)$-VMA problems. We show now that these problems are NP-hard. We first prove the following lemma.

Lemma 15. *Given an instance of the VMA problem, any solution $\pi = \{A_1, \ldots, A_m\}$ where $\ell(A_i) \neq x^*$ for some $i \in [1, m] : A_i \neq \emptyset$, has power consumption $P(\pi) > \rho^* \ell(D) = \rho^* \sum_{d \in D} \ell(d)$.*

Proof: The total cost of π is $P(\pi) = \sum_{i \in [1,m]} f(\ell(A_i))$ which, from Observation 2, satisfies

$$\begin{aligned} P(\pi) &> \sum_{i \in [1,m] : A_i \neq \emptyset} \ell(A_i)\rho^* \\ &= \rho^* \sum_{i \in [1,m] : A_i \neq \emptyset} \sum_{d \in A_i} \ell(d) = \rho^* \sum_{d \in D} \ell(d). \end{aligned}$$

■

We show now in the following theorem that the different versions of the (C, m)-VMA problem with unbounded C or m are NP-hard.

Theorem 12. *The $(C, \cdot)$-VMA, $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems are strongly NP-hard, even if α is constant.*

Proof: We show a reduction from 3-Partition defined as follows [75], which is strongly NP-complete.

INSTANCE: Set A of $3k$ elements, a bound $B \in \mathbb{Z}^+$ and, for each $a \in A$, a size $s(a) \in \mathbb{Z}^+$ such that $B/4 < s(a) < B/2$ and $\sum_{a \in A} s(a) = kB$.

QUESTION: can A be partitioned into k disjoint sets $\{A_1, A_2, \ldots, A_k\}$ such that $\sum_{a \in A_i} s(a) = B$ for each $1 \leq i \leq k$?

The reduction is as follows. Given an instance of 3-Partition on a set $A = \{a_1, \ldots, a_{3k}\}$ with bound B, and given a fixed value $\alpha > 1$, we define an instance $\mathcal{I}$ of $(\cdot, \cdot)$-VMA as follows: $D = \{a_1, \ldots, a_{3k}\}$, $\ell(\cdot) = s(\cdot)$, and $b = B^\alpha(\alpha - 1)$ (i.e., $x^* = B$). (For the proof of the $(C, \cdot)$-VMA and $(\cdot, m)$-VMA problems it is enough to set $C = B$ and $m = k$ when required.) We show now that the answer to the 3-Partition problem is YES if and only if the output $\pi = \{A_1, A_2, \ldots, A_m\}$ of the $(\cdot, \cdot)$-VMA problem on input $\mathcal{I}$ is such that $\sum_{i=1}^m f(\ell(A_i)) = kf(B)$.

For the direct implication, assume that there exists a partition $\{A_1, A_2, \ldots, A_k\}$ of A such that for each $i \in [1, k]$, $\sum_{a \in A_i} s(a) = B$. Then, in the context of the $(\cdot, \cdot)$-VMA problem, such partition has cost $\sum_{i=1}^m f(\ell(A_i)) = kf(B)$. We claim that any partition has at least cost $kf(B)$. In order to prove it, assume for the sake of contradiction that there is a partition $\pi' = \{A'_1, A'_2, \ldots, A'_m\}$ of $(\cdot, \cdot)$-VMA on input $\mathcal{I}$ with cost less than $kf(B)$. Then, there is some $i \in [1, m]$ such that $A'_i \neq \emptyset$ and $\ell(A'_i) \neq B$. From Lemma 15, $P(\pi') > \rho^* \ell(D) = (f(x^*)/x^*)kB$. Since $B = x^*$, we have that $P(\pi') > kf(B)$, which is a contradiction.

To prove the reverse implication, assume an output $\pi = \{A_1, A_2, \ldots, A_m\}$ of the $(\cdot, \cdot)$-VMA problem on input $\mathcal{I}$ such that $P(\pi) = \sum_{i=1}^m f(\ell(A_i)) = kf(B)$. Then, it must be $\forall i \in [1, m]$: $A_i \neq \emptyset, \ell(A_i) = B$. Otherwise, from Lemma 15, $P(\pi) > kf(B)$, a contradiction. ■

It is known that strongly NP-hard problems cannot have a fully polynomial-time approximation scheme (FPTAS) [158]. Hence, the following corollary.

Corollary 15. *The $(C, \cdot)$-VMA, $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems do not have fully polynomial-time approximation schemes (FPTAS), even if α is constant.*

In the following sections we show that, while the $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems have polynomial-time approximation schemes (PTAS), the $(C, \cdot)$-VMA problem cannot be approximated below $\frac{3}{2} \cdot \frac{\alpha - 1 + (2/3)^\alpha}{\alpha}$.

7.3.2 The $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA Problems Have PTAS

We have proved that the $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems are NP-hard in the strong sense and that, hence, there exists no FPTAS for them. However, Alon *et al.* [12], proved that if a function $f(\cdot)$ satisfies a condition denoted $F*$, then the problem of scheduling jobs in m identical machines so that $\sum_i f(M_i)$ is minimized has a PTAS, where M_i is the load of the jobs allocated to machine i. This result implies that if our function $f(\cdot)$ satisfies condition $F*$, the same PTAS can be used for the $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems. From Observation 6.1 in [66], it can be derived that, in fact, our power consumption function $f(\cdot)$ satisfies condition $F*$. Hence, the following theorem.

Theorem 13. *There are polynomial-time approximation schemes (PTAS) for the $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems.*

7.3.3 Bounds on the Approximability of the $(C, \cdot)$-VMA Problem

We study now the $(C, \cdot)$-VMA problem, where we consider an unbounded number of machines with bounded capacity C. We will provide a lower bound on its approximation ratio, independently on the relation between x^* and C; and upper bounds for the cases when $x^* \geq C$ and $x^* < C$.

7.3.3.1 Lower bound on the approximation ratio

The following theorem shows a lower bound on the approximation ratio of any offline algorithm for $(C, \cdot)$-VMA.

Theorem 14. *No algorithm achieves an approximation ratio smaller than $\frac{3}{2} \cdot \frac{\alpha - 1 + (\frac{2}{3})^\alpha}{\alpha}$ for the $(C, \cdot)$-VMA problem unless $\mathrm{P} = \mathrm{NP}$.*

Proof: The claim is proved showing a reduction from the partition problem [75]. In the partition problem there is a set $A = \{a_1, a_2, \ldots, a_n\}$ of n elements, there is a size $s(a)$ for each element $a \in A$, and the sum $M = \sum_{a \in A} s(a)$ of the sizes of the elements in A. The problem decides whether there is a subset $A' \subset A$ such that $\sum_{a \in A'} s(a) = M/2$.

From an instance of the partition problem, we construct an instance of the $(C, \cdot)$-VMA problem as follows. The set of VMs in the system is $D = \{a_1, a_2, \ldots, a_n\}$, the load function is $\ell(\cdot) = s(\cdot)$, the capacity of each PM is set to $C = M/2$, and b is set to $b = C^\alpha(\alpha - 1)$ (i.e., $x^* = C$). Let us study the optimal partition π^* such that the total power consumption $P(\pi^*)$ is minimized. If there is a partition of D such that each subset in this partition has load $M/2$ then, from Observation 2, π^* has all the VMs assigned to two PMs. Otherwise, π^* needs at least 3 PMs to allocate all the VMs. From Corollary 14, the power consumption of this solution is lower bounded by the power of a (maybe unfeasible) partition that balances the load among the 3 PMs as evenly as possible. Formally,

$$\exists A' : \sum_{a \in A'} s(a) = M/2 \quad \Rightarrow \quad P(\pi^*) = 2b + 2\left(\frac{M}{2}\right)^\alpha = 2b + 2C^\alpha$$

$$\nexists A' : \sum_{a \in A'} s(a) = M/2 \quad \Rightarrow \quad P(\pi^*) \geq 3b + 3\left(\frac{M}{3}\right)^\alpha = 3b + 3\left(\frac{2C}{3}\right)^\alpha .$$

Comparing both values we obtain the following ratio.

$$\begin{aligned}\rho &= \frac{3b + 3\left(\frac{2C}{3}\right)^\alpha}{2b + 2C^\alpha} \\ &= \frac{3C^\alpha(\alpha-1) + 3\left(\frac{2C}{3}\right)^\alpha}{2C^\alpha(\alpha-1) + 2C^\alpha} \\ &= \frac{3}{2}\cdot\frac{\alpha - 1 + (\frac{2}{3})^\alpha}{\alpha}.\end{aligned}$$

Therefore, given any $\epsilon > 0$, having a polynomial-time algorithm $\mathcal{A}$ with approximation ratio $\rho - \epsilon$ would imply that this algorithm could be used to decide if there is a subset $A' \subset A$ such that $\sum_{a\in A'} s(a) = M/2$. In other words, this algorithm would be able to solve the partition problem. This contradicts the fact that the partition problem is NP-hard and no polynomial time algorithm solves it unless P $=$ NP. Therefore, there is no algorithm that achieves a $\rho - \epsilon = \frac{3}{2}\cdot\frac{\alpha-1+(\frac{2}{3})^\alpha}{\alpha} - \epsilon$ approximation ratio for the $(C,\cdot)$-VMA problem unless P $=$ NP. ■

7.3.3.2 Upper bound on the approximation ratio for $x^* \geq C$

We study now an upper bound on the competitive ratio of the $(C,\cdot)$-VMA problem for the case when $x^* \geq C$. Under this condition, the best is to load each PM to its full capacity. Intuitively, an optimal solution should load every machine up to its maximum capacity or, if not possible, should balance the load among PMs to maximize the average load. The following lemma formalizes this observation.

Lemma 16. *For any system with unbounded number of PMs where $x^* \geq C$ the power consumption of the optimal assignment π^* is lower bounded by the power consumption of a (possibly not feasible) solution where $\ell(D)$ is evenly distributed among $\overline{m}$ PMs, where $\overline{m}$ is the minimum number of PMs required to allocate all VMs (i.e., the optimal solution of the packing problem). That is,*

$$P(\pi^*) \geq \overline{m}\cdot b + \overline{m}(\ell(D)/\overline{m})^\alpha.$$

Proof: Denote the number of PMs used in an optimal $(C,\cdot)$-VMA solution π^* by m^*. By Corollary 14, we know that

$$P(\pi^*) \geq m^* b + m^*(\ell(D)/m^*)^\alpha.$$

Given that $\overline{m} \leq m^*$, we know that $\ell(D)/m^* \leq \ell(D)/\overline{m} \leq C \leq x^*$. Thus, for evenly-balanced loads the power consumption is reduced if the number of PMs is reduced, that is

$$m^* b + m^*(\ell(D)/m^*)^\alpha \geq \overline{m}\cdot b + \overline{m}(\ell(D)/\overline{m})^\alpha.$$

Hence, the claim follows. ■

Now we prove an upper bound on the approximation ratio showing a reduction to bin packing [75]. The reduction works as follows. Let each PM be seen as a bin of capacity C, and each VM be seen as an object to be placed in the bins, whose size is the VM load. Then, a solution for this bin packing problem instance yields a feasible (perhaps suboptimal) solution for the instance of $(C,\cdot)$-VMA. Moreover, using any bin-packing approximation algorithm, we obtain a feasible solution for $(C,\cdot)$-VMA that approximates the minimal number of PMs used. The power consumption of this solution approximates the power consumption of the optimal solution π^* of the instance of $(C,\cdot)$-VMA. In order to compute an upper bound on the approximation ratio of this algorithm, we will compare the power consumption of such solution against a lower bound on the power consumption of π^*. The following theorem shows the approximation ratio obtained.

Theorem 15. *For every $\epsilon > 0$, there exists an approximation algorithm for the $(C,\cdot)$-VMA problem when $x^* \geq C$ that achieves an approximation ratio of*

$$\rho < 1 + \epsilon + \frac{C^\alpha}{b} + \frac{1}{\overline{m}},$$

where $\overline{m}$ is the minimum number of PMs required to allocate all the VMs.

Proof: Consider an instance of the $(C,\cdot)$-VMA problem. If $\ell(D) \leq C$, the optimal solution is to place all the VMs in one single PM. Hence, we assume in the rest of the proof that $\ell(D) > C$. Define the corresponding instance of bin packing following the reduction described above. Let the optimal number of bins to accommodate all VMs be $\overline{m}$. As shown in [69], for every $\epsilon > 0$, there is a polynomial-time algorithm that fits all VMs in $\widehat{m}$ bins, where $\widehat{m} \leq (1+\epsilon)\overline{m} + 1$. From Lemma 14, once the number of PMs used $\widehat{m}$ is fixed, the power consumption is maximized when the load is unbalanced to the maximum. I.e., the power consumption of the assignment is at most $\widehat{m}b + (\ell(D)/C)C^\alpha$. On the other hand, as shown in Lemma 16, the power consumption of the optimal $(C,\cdot)$-VMA solution is at least $\overline{m}\cdot b + \overline{m}\left(\frac{\ell(D)}{\overline{m}}\right)^\alpha$. Then, we compute a bound on the approximation ratio as follows.

$$\rho \leq \frac{\widehat{m}b + \left(\frac{\ell(D)}{C}\right)C^\alpha}{\overline{m}\cdot b + \overline{m}\left(\frac{\ell(D)}{\overline{m}}\right)^\alpha} \tag{7.1}$$

$$< \frac{\widehat{m}b + \left(\frac{\ell(D)}{C}\right)C^\alpha}{\overline{m}\cdot b + \overline{m}\left(\frac{C}{2}\right)^\alpha}, \tag{7.2}$$

where the second inequality comes from $\ell(D)/\overline{m} > C/2$. (If $\ell(D)/\overline{m} \leq C/2$, there must be two PMs whose loads add up to less than C, which contradicts the fact that $\overline{m}$ is the number of bins used in the optimal solution of bin packing.) Let $\gamma = (x^*/C)^\alpha$. Then, replacing $b = \gamma C^\alpha(\alpha - 1)$, in Eq. (7.1) we have

$$
\begin{aligned}
\rho &< \frac{\widehat{m}\gamma C^\alpha(\alpha-1) + \left(\frac{\ell(D)}{C}\right) C^\alpha}{\overline{m}\gamma C^\alpha(\alpha-1) + \overline{m}\left(\frac{C}{2}\right)^\alpha} \\
&= \frac{\widehat{m}\gamma(\alpha-1) + \left(\frac{\ell(D)}{C}\right)}{\overline{m}\gamma(\alpha-1) + \overline{m}\left(\frac{1}{2}\right)^\alpha} \le \frac{\widehat{m}\gamma(\alpha-1) + \overline{m}}{\overline{m}\gamma(\alpha-1) + \left(\frac{\overline{m}}{2^\alpha}\right)} &\quad (7.3) \\
&\le \frac{(\overline{m}(1+\epsilon)+1))\gamma(\alpha-1) + \overline{m}}{\overline{m}\gamma(\alpha-1) + \left(\frac{\overline{m}}{2^\alpha}\right)} &\quad (7.4) \\
&= \frac{(1+\epsilon)\gamma(\alpha-1)+1}{\gamma(\alpha-1) + \left(\frac{1}{2^\alpha}\right)} + \frac{\gamma(\alpha-1)}{\overline{m}\gamma(\alpha-1) + \left(\frac{\overline{m}}{2^\alpha}\right)} \\
&= \frac{2^\alpha((1+\epsilon)\gamma(\alpha-1)+1)}{2^\alpha\gamma(\alpha-1)+1} + \frac{2^\alpha\gamma(\alpha-1)}{\overline{m}(2^\alpha\gamma(\alpha-1)+1)} \\
&< \frac{(1+\epsilon)\gamma(\alpha-1)+1}{\gamma(\alpha-1)} + \frac{1}{\overline{m}} \\
&= 1+\epsilon+\frac{1}{\gamma(\alpha-1)} + \frac{1}{\overline{m}} = 1+\epsilon+\frac{C^\alpha}{b} + \frac{1}{\overline{m}}
\end{aligned}
$$

Inequality (7.3) follows from $\ell(D)/C \le \overline{m}$, Inequality (7.4) from the approximation algorithm for bin packing, and the last inequality is because $\overline{m} > 0$. ■

7.3.3.3 Upper bound on the approximation ratio for $x^* < C$

We study now the $(C,\cdot)$-VMA problem when $x^* < C$. In this case, the optimal load per PM is less than its capacity, so an optimal solution would load every PM to x^* if possible, or try to balance the load close to x^*. In this case we slightly modify the bin packing algorithm described above, reducing the bin size from C to x^*. Then, using an approximation algorithm for this bin packing problem, the following theorem can be shown.

Theorem 16. *For every $\epsilon > 0$, there exists an approximation algorithm for the $(C,\cdot)$-VMA problem when $x^* < C$ that achieves an approximation ratio of*

$$\rho < \frac{\overline{m}}{m^*}\left((1+\epsilon) + \frac{1}{\alpha-1}\right) + \frac{1}{m^*},$$

where m^* is the number of PMs used by the optimal solution of $(C,\cdot)$-VMA, and $\overline{m}$ is the minimum number of PMs required to allocate all the VMs without exceeding load x^* (i.e., the optimal solution of the bin packing problem).

Proof: Consider an instance of the $(C,\cdot)$-VMA problem. If $\ell(D) \le x^*$ then the optimal solution is to assign all the VMs to one single PM. Then, in the rest of the proof we assume that $\ell(D) > x^*$. Assuming m^* to be the number of PMs of an optimal $(C,\cdot)$-VMA solution π^* for load $\ell(D)$, from Corollary 14, we can claim that the power consumption $P(\pi^*)$ can be bounded as $P(\pi^*) \ge m^*b + m^*(\ell(D)/m^*)^\alpha$.

Now, let $\overline{m}$ be the minimum number of PMs required to allocate all the VMs of the $(C,\cdot)$-VMA problem without exceeding load x^*. As shown in [69], for every $\epsilon > 0$, there is a polynomial-time algorithm that fits all VMs in $\widehat{m}$ bins, where $\widehat{m} \leq (1+\epsilon)\overline{m}+1$. From Lemma 14, this approximation results in a power consumption no larger than $\widehat{m}b + (\ell(D)/x^*)(x^*)^\alpha$. Hence, the approximation ratio ρ of the solution obtained wit this algorithm can be bounded as follows.

$$\rho \leq \frac{\widehat{m}b + \left(\frac{\ell(D)}{x^*}\right)(x^*)^\alpha}{m^*b + m^*\left(\frac{\ell(D)}{m^*}\right)^\alpha}. \tag{7.5}$$

Since $\ell(D) > x^*$, we know that $\ell(D)/m^* > x^*/2$, since otherwise there are two used PMs whose load is no larger than x^*, contradicting by Lemma 13 the definition of m^*. Also, from the definition of $\overline{m}$, it follows that $\ell(D) \leq \overline{m} \cdot x^*$. Finally, recall that $b = (x^*)^\alpha(\alpha-1)$. Applying these results to Eq. (7.5) we have the following.

$$\begin{aligned}
\rho &< \frac{\widehat{m}(x^*)^\alpha(\alpha-1) + \left(\frac{x^*\overline{m}}{x^*}\right)(x^*)^\alpha}{m^*(x^*)^\alpha(\alpha-1) + m^*\left(\frac{x^*}{2}\right)^\alpha} \\
&= \frac{\widehat{m}(\alpha-1) + \overline{m}}{m^*(\alpha-1) + m^*\left(\frac{1}{2}\right)^\alpha} \leq \frac{(\overline{m}(1+\epsilon)+1)(\alpha-1) + \overline{m}}{m^*(\alpha-1) + \frac{m^*}{2^\alpha}} \\
&= \frac{\overline{m}(1+\epsilon)(\alpha-1) + \overline{m}}{m^*(\alpha-1) + \frac{m^*}{2^\alpha}} + \frac{\alpha-1}{m^*(\alpha-1) + \frac{m}{2^\alpha}} \\
&= \frac{\overline{m}}{m^*}\frac{2^\alpha((1+\epsilon)(\alpha-1)+1)}{2^\alpha(\alpha-1)+1} + \frac{2^\alpha(\alpha-1)}{2^\alpha m^*(\alpha-1) + m^*} \\
&\leq \frac{\overline{m}}{m^*}\left((1+\epsilon) + \frac{1}{\alpha-1}\right) + \frac{1}{m^*},
\end{aligned}$$

where the first inequality comes from applying the results aforementioned, and second one from using $\widehat{m} = \overline{m}(1+\epsilon)+1$, while the last one results from simplifying the previous equation. ■

7.4 Online Analysis

In this section, we study the online version of the VMA problem, i.e., when the VMs are revealed one by one. We first study lower bounds and then provide online algorithms and prove upper bounds on their competitive ratio.

7.4.1 Lower Bounds

In this section, we compute lower bounds on the competitive ratio for $(\cdot,\cdot)$-VMA, $(C,\cdot)$-VMA, $(\cdot,m)$-VMA, (C,m)-VMA and $(\cdot,2)$-VMA problems. We start with one general construction that is used to obtain lower bounds on the first four cases. Then, we develop special constructions for $(\cdot, m)$-VMA and $(\cdot, 2)$-VMA that improve the lower bounds for these two problems.

7.4.1.1 General Construction

We prove lower bounds on the competitive ratio of $(\cdot,\cdot)$-VMA, $(C,\cdot)$-VMA, $(\cdot,m)$-VMA and (C,m)-VMA problems. These lower bounds are shown in the following two theorems. In Theorem 17, we prove a lower bound on the competitive ratio that is valid in the cases when C is unbounded and when it is larger or equal than x^*. The case $C \leq x^*$ is covered in Theorem 18.

Theorem 17. *There exists an instance of problems $(\cdot,\cdot)$-VMA, $(\cdot,m)$-VMA, $(C,\cdot)$-VMA and (C,m)-VMA when $C > x^*$, such that no online algorithm can guarantee a competitive ratio smaller than $\frac{(3/2)2^\alpha-1}{2^\alpha-1}$.*

Proof: We consider a scenario where, for any online algorithm, an adversary injects VMs of size ϵx^* ($\epsilon > 0$ is an arbitrarily small constant) to the system until the algorithm starts up a new PM. Let us assume that the total number of VMs injected is k. According to the adversary's behavior, the assignment of the VMs should be that all the VMs except one are allocated to a single PM while the second PM has only one VM. Depending on what the optimal solution is, we discuss the following two cases:
Case 1: $k \leq \frac{1}{\epsilon}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha}$. The optimal solution will allocate all the VMs to a single PM. Consequently, the competitive ratio of the online algorithm satisfies

$$\rho(k) \geq \lim_{\epsilon \to 0}\left(\frac{((k-1)\epsilon x^*)^\alpha + (\epsilon x^*)^\alpha + 2b}{(k\epsilon x^*)^\alpha + b}\right).$$

It can be easily verified that function $\rho(k)$ is monotone decreasing with k. That is, $\rho(k)$ is minimized when $k = \frac{1}{\epsilon}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha}$. As a result, we obtain,

$$\begin{aligned}
\rho(k) &\geq \lim_{\epsilon \to 0}\left(\frac{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + (\epsilon x^*)^\alpha + 2b}{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + b}\right) \\
&= \frac{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + 2(x^*)^\alpha(\alpha-1)}{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + (x^*)^\alpha(\alpha-1)} \\
&= \frac{3-2^{1-\alpha}}{2-2^{1-\alpha}} = \frac{(3/2)2^\alpha-1}{2^\alpha-1}.
\end{aligned}$$

Case 2: $k > \frac{1}{\epsilon}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha}$. The optimal solution will use two PMs with $k/2$ PMs assigned to each PM. Accordingly, the competitive ratio of the online algorithm satisfies

$$\rho(k) \geq \lim_{\epsilon \to 0} \left(\frac{((k-1)\epsilon x^*)^\alpha + (\epsilon x^*)^\alpha + 2b}{2\left(\frac{k\epsilon x^*}{2}\right)^\alpha + 2b} \right).$$

Similarly, we observe that $\rho(k)$ is monotone increasing with k. Consequently, the following inequality applies.

$$\begin{aligned}\rho(k) &\geq \lim_{\epsilon \to 0} \left(\frac{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + (\epsilon x^*)^\alpha + 2b}{2\left(\frac{1}{2}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + 2b} \right) \\ &= \frac{\left(\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + 2(x^*)^\alpha(\alpha-1)}{2\left(\frac{1}{2}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*\right)^\alpha + 2(x^*)^\alpha(\alpha-1)} \\ &= \frac{3-2^{1-\alpha}}{2-2^{1-\alpha}} = \frac{(3/2)2^\alpha - 1}{2^\alpha - 1}\end{aligned}$$

Note that it can also happen that $C < \left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha} x^*$. In this case, k is smaller than $\frac{1}{\epsilon}\left(\frac{\alpha-1}{1-2^{1-\alpha}}\right)^{1/\alpha}$. Therefore, the competitive ratio is always larger than $\frac{(3/2)2^\alpha-1}{2^\alpha-1}$, proving the lower bound. ■

Theorem 18. *There exists an instance of problems $(C,\cdot)$-VMA and (C,m)-VMA when $C \leq x^*$ such that no online algorithm can guarantee a competitive ratio smaller than $(C^\alpha + 2b)/(b + \max(C^\alpha, 2(C/2)^\alpha + b))$.*

Proof: Similarly to the proof of Theorem 17, we prove the result by considering an adversarial injection of VMs of size ϵC. This injection stops when a new PM started up by an online algorithm. We discuss the following two cases:

Case 1: $k \leq 1/\epsilon$. In this case, the optimal algorithm will assign all the VMs to a single PM. The competitive ratio of the online algorithm satisfies

$$\begin{aligned}\rho(k) &\geq \lim_{\epsilon \to 0} \frac{((k-1)\epsilon C)^\alpha + (\epsilon C)^\alpha + 2b}{(k\epsilon C)^\alpha + b} \\ &\geq \lim_{\epsilon \to 0} \frac{(1-\epsilon)^\alpha C^\alpha + 2b}{C^\alpha + b} \\ &\geq \frac{C^\alpha + 2b}{C^\alpha + b} \geq 2 - \frac{1}{\alpha}\end{aligned}$$

The second inequality results from applying $k \leq 1/\epsilon$, which is observed from the monotone decreasing property of function $\rho(k)$. The last inequality comes from computing the limit when ϵ goes to 0 and by applying $b \geq C^\alpha(\alpha - 1)$.

Case 2: $k > 1/\epsilon$. In this case, the adversary stops injecting VMs as there will be, mandatorily, two active PMs, one of them not capable to allocate more VMs and the second one hosting one single VM. Since all the VMs can not be consolidated to a single PM. The optimal solution would use also two PMs but evenly balancing the loads among them. The competitive ratio of the online algorithm satisfies

$$\begin{aligned}\rho(k) &= \lim_{\epsilon\to 0}\left(\frac{((k-1)\epsilon C)^\alpha + (\epsilon C)^\alpha + 2b}{2\left(\frac{k\epsilon C}{2}\right)^\alpha + 2b}\right)\\ &= \lim_{\epsilon\to 0}\left(\frac{C^\alpha + (\epsilon C)^\alpha + 2b}{2\left(\frac{C+\epsilon C}{2}\right)^\alpha + 2b}\right)\\ &= \frac{C^\alpha + 2b}{2\left(\frac{C}{2}\right)^\alpha + 2b}.\end{aligned}$$

Hence, combining the results from both cases 1 and 2 we obtain the bound presented in Theorem 18. ■

7.4.1.2 Special Constructions for $(\cdot, m)$-VMA and $(\cdot, 2)$-VMA

We show first that for m PMs there is a lower bound on the competitive ratio that improves the previous lower bound when $\alpha > 4.5$. Secondly, we prove a particular lower bound for problem $(\cdot, 2)$-VMA, that improves the previous lower bound when $\alpha > 3$.

Theorem 19. *There exists an instance of problem $(\cdot, m)$-VMA such that no online algorithm can guarantee a competitive ratio smaller than $3^\alpha/(2^{\alpha+2}+\epsilon)$ for any $\epsilon > 0$.*

Proof: We prove the result by giving an adversarial arrival of VMs. We evaluate the competitive ratio of any online algorithm ALG with respect to an algorithm OPT that distributes the VMs among all the PMs "as evenly as possible". We define a value $\beta > 1$ such that $\epsilon \geq (\alpha-1)/\beta^\alpha$ for some value $\epsilon > 0$. Note that such value β can be defined for any $\epsilon > 0$. The adversarial arrival follows. In a first phase, m VMs arrive, each with load βx^*.

Let π be the partition given by ALG. We show first that if π uses less than $3m/4$ PMs[2] or some PM is assigned more than 2 VMs there exists another partition that can be obtained from π, it uses exactly $3m/4$ PMs, no PM is assigned more than 2 VMs, and the power consumption is not worse.

If π uses less than $3m/4$ PMs, then there exists another partition π' that uses exactly $3m/4$ PMs with a power consumption that is not worse than $P(\pi)$. To see why, notice that there are PMs in π that are assigned more than one VM and that each load is $\beta x^* > x^*$. Then, applying repeatedly Lemma 13 until $3m/4$ PMs are used, where ℓ_1 and ℓ_2 are the loads of any pair of VMs assigned to the same PM, a partition π' such that $P(\pi') \leq P(\pi)$ can be obtained.

If in π' some PM is assigned more than 2 VMs, then there exists another partition π'' where no PM is assigned more than 2 VMs with a power consumption that is not worse than $P(\pi')$. To

[2]For clarity we omit floors and ceilings in the proof.

see why, consider the following reassignment procedure. Repeatedly until there is no such PM, locate a PM s_i with at least 3 VMs. Then, locate a PM s_j with one single VM (which exists by the pigeonhole principle). Then, move one VM from s_i to s_j. From Lemma 14 each movement decreases the power consumed. Hence, π'' is still a partition that uses $3m/4$ PMs, each PM has at most 2 VMs assigned, and $P(\pi'') \leq P(\pi')$.

Then, we know that $P(\pi)$ is not smaller than the power consumption of a partition where exactly $3m/4$ PMs are used and no PM is assigned more than 2 VMs. On the other hand, OPT would have assigned each VM to a different PM. Thus, using that $x^* = (b/(\alpha-1))^{1/\alpha}$, the competitive ratio is

$$\begin{aligned}\rho &\geq \frac{(2\beta x^*)^\alpha m/4 + (\beta x^*)^\alpha m/2 + 3mb/4}{m(\beta x^*)^\alpha + mb}\\ &\geq \frac{(2^{\alpha-2}+1/2)\beta^\alpha}{\beta^\alpha + (\alpha-1)} \geq 2^{\alpha-3} + 1/4,\end{aligned}$$

where the last inequality follows from $\beta^\alpha \geq (\alpha-1)$. Finally, observe that $2^{\alpha-3}+1/4 \geq 3^\alpha/(2^{\alpha+2}+\epsilon)$ for $\alpha > 1$. No more VMs arrive in this case.

Let us consider now the the case where ALG assigns the m initial VMs to more than $3m/4$ PMs. Then, after ALG has assigned the first m VMs, a second batch of $m/2$ VMs arrive, each VM with load $2\beta x^*$. Let π be the partition output by ALG after this second batch is assigned. If in π two of the second batch VMs are assigned to the same PM s_i, by the pigeonhole principle there is at least one PM s_j with at most load βx^*. Then, from Lemma 14, the power consumed is reduced if one of the new VMs is moved from s_i to s_j. After repeating this process as many times as possible, a partition π' is obtained where each of the VMs of the second batch is assigned to a different PM, and $P(\pi') \leq P(\pi)$. Since ALG used more than $3m/4$ PMs in the first batch, in π', there are at least $m/4$ PMs with load $3\beta x^*$. On the other hand, OPT can distribute all the VMs in such a way that each PM has a load of $2\beta x^*$. Thus, the bound on the competitive ratio is as follows.

$$\rho \geq \frac{m(3\beta x^*)^\alpha/4}{m(2\beta x^*)^\alpha + mb} \geq \frac{3^\alpha}{2^{\alpha+2}+\epsilon},$$

where the last inequality follows from $\epsilon \geq (\alpha-1)/\beta^\alpha$. ■

Now, we show a stronger lower bound on the competitive ratio for $(\cdot, 2)$-VMA problem.

Theorem 20. *There exists an instance of problem* $(\cdot,2)$*-VMA such that no online algorithm can guarantee a competitive ratio smaller than* $3^\alpha/2^{\alpha+1}$.

Proof: We prove the result by showing an adversarial arrival of VM. We evaluate the competitive ratio of any online algorithm ALG with respect to an optimal algorithm OPT that knows the future VM arrivals. The adversarial arrival follows. In a first phase two VM d_1 and d_2 arrive, with loads $\ell(d_1) = \ell(d_2) = 6x^*$ (Recall from Section 7.2 that $x^* = (b/(\alpha-1))^{1/\alpha}$).

If ALG assigns both VMs to the same PM, the power consumed will be $(12x^*)^\alpha + b$, whereas OPT would assign them to different PMs, with a power consumption of $2((6x^*)^\alpha + b)$. Hence, the ratio ρ would be

$$\begin{aligned}\rho &= \frac{(12x^*)^\alpha + b}{2((6x^*)^\alpha + b)} > \frac{12^\alpha}{2(6^\alpha + \alpha - 1)} \\ &> \frac{12^\alpha}{2(6^\alpha + 2^\alpha)} = \frac{6^\alpha}{2(3^\alpha + 1)},\end{aligned}$$

where the first inequality follows from $\alpha > 1$ and the second from $\alpha - 1 < 2^\alpha$ for any $\alpha > 1$. It is enough to prove that $6^\alpha/(2(3^\alpha + 1)) \geq (3/2)^\alpha/2$, or equivalently $4^\alpha \geq 3^\alpha + 1$, which is true for any $\alpha > 1$. Then, there are no new VM arrivals.

If, otherwise, ALG assigns each VM d_1 and d_2 to a different PM, then a third VM d_3 arrives, with load $\ell(d_3) = 12x^*$. Then, ALG must assign it to one of the PMs. Independently of which PM is used, the power consumption of the final configuration is $(18x^*)^\alpha + (6x^*)^\alpha + 2b$. On its side, OPT assigns d_1 and d_2 to one PM, and d_3 to the other, with a power consumption of $2((12x^*)^\alpha + b)$. Hence, the competitive ratio ρ is

$$\begin{aligned}\rho &= \frac{(18x^*)^\alpha + (6x^*)^\alpha + 2b}{2((12x^*)^\alpha + b)} > \frac{18^\alpha + 6^\alpha}{2(12^\alpha + \alpha - 1)} \\ &> \frac{18^\alpha + 6^\alpha}{2(12^\alpha + 4^\alpha)} \geq (3/2)^\alpha/2,\end{aligned}$$

where the first inequality follows from $\alpha > 1$, the second from $\alpha - 1 < 4^\alpha$ for any $\alpha > 1$, and the third from $(9^\alpha + 3^\alpha)/(6^\alpha + 2^\alpha) \geq (3/2)^\alpha$, what can be checked to be true. Then, there are no new VM arrivals and the claim follows. ■

7.4.2 Upper Bounds

Now, we study upper bounds for $(\cdot,\cdot)$-VMA, $(C,\cdot)$-VMA, and $(\cdot,2)$-VMA problems. We start giving two online VMA algorithms, that we denote as *VMA1* and *VMA2* and that can be found in Algorithm 2 and Algorithm 3. We study algorithm *VMA1* for both $(\cdot,\cdot)$-VMA and $(C,\cdot)$-VMA problems while *VMA2* is only studied for the $(\cdot,\cdot)$-VMA case. These algorithms use the load of the new revealed VM in order to decide the PM where it will be assigned. The behaviour of both algorithms is similar. In algorithm *VMA1*, if the load of the revealed VM is strictly larger than $\min\{x^*, C\}/2$, the algorithm assigns this VM to a new PM without any other VM already assigned to it. Otherwise, the algorithm schedules the revealed VM to any loaded PM whose current load is smaller or equal than $\frac{\min\{x^*,C\}}{2}$. Hence, when this new VM is assigned, the load of this PM remains smaller than $\min\{x^*, C\}$. If there is no such loaded PM, the revealed VM is assigned to a new PM. On the other hand, algorithm *VMA2* shares the same behaviour but uses a different threshold. In this case, if the load of the revealed VM is strictly larger than $\min\{x^*, C\}$, the algorithm assigns this VM to a new PM. If the load is smaller than x^*, *VMA2* schedules the new VM to the most loaded PM whose load is smaller than x^*.

Note that, since the case under consideration assumes the existence of an unbounded number of PMs, there exists always one new PM. As mentioned, a detailed description of these algorithms is shown in Algorithm 2 and Algorithm 3. As before, A_j denotes the set of VMs assigned to PM s_j at a given time.

Algorithm 2: Online algorithm *VMA1* for $(\cdot,\cdot)$-VMA and $(C,\cdot)$-VMA problems.

```
for each VM $d_i$ do
    if $\ell(d_i) > \frac{\min\{x^*,C\}}{2}$ then
        $d_i$ is assigned to a new PM
    else
        $d_i$ is assigned to any loaded PM $s_j$ where $\ell(A_j) \leq \frac{\min\{x^*,C\}}{2}$. If such loaded PM
        does not exist, $d_i$ is assigned to a new PM.
```

Algorithm 3: Online algorithm *VMA2* for the $(\cdot,\cdot)$-VMA problem.

```
for each VM $d_i$ do
    if $\ell(d_i) \geq x^*$ then
        $d_i$ is assigned to a new PM
    else
        $d_i$ is assigned to the PM $s_j$ such that $\ell(A_k) \leq \ell(A_j) < x^*$ for all $k$. If such loaded
        PM does not exist, $d_i$ is assigned to a new PM.
```

7.4.2.1 Upper Bounds for the $(\cdot,\cdot)$-VMA and $(C,\cdot)$-VMA problems

We prove the approximation ratio of Algorithm 2 in the following two theorems.

Theorem 21. *There exists an online algorithm for $(\cdot,\cdot)$-VMA and $(C,\cdot)$-VMA when $x^* < C$ that achieves the following competitive ratio:*

$$\rho = 1, \text{ if no VM } d_i \text{ has load such that } \ell(d_i) < x^*,$$
$$\rho \leq \left(1 - \tfrac{1}{\alpha}\left(1 - \tfrac{1}{2^\alpha}\right)\right)\left(2 + \tfrac{x^*}{\ell(D_s)}\right), \text{ otherwise.}$$

Proof: We proceed with the analysis of the competitive ratio of Algorithm 2 shown above. Let us first consider an optimal algorithm, that is, an algorithm that gives an optimal solution for any instance. Let us denote by π^* the optimal solution obtained by the optimal algorithm, and A_i the load assigned to PM s_i in that solution, for a particular instance of VMA problem. Furthermore, load A_i is decomposed in $d_{i_1}, d_{i_2}, \ldots, d_{i_{k_i}}$, where each d_{i_j} is a VM that π^* assigns to s_i. Using simple algebra, it holds:

$$f(\ell(A_i)) = \frac{f(\ell(A_i))}{\ell(A_i)}(\ell(d_{i_1}) + \ell(d_{i_2}) + \cdots + \ell(d_{i_{k_i}})).$$

It is possible now to split the set A_i in two sets, one with those VMs assigned to s_i whose load is strictly smaller than x^* and a second set that contains those VMs assigned to s_i whose load is bigger than x^*. In terms of notation, we say that A_i is split in B_i and S_i (where B stands for Big loads and S stands for Small loads). Therefore, it also holds:

$$f(\ell(A_i)) = \sum_{d_{i_j} \in B_i} \frac{f(\ell(A_i))}{\ell(A_i)} \ell(d_{i_j}) + \sum_{d_{i_j} \in S_i} \frac{f(\ell(A_i))}{\ell(A_i)} \ell(d_{i_j}).$$

On the other hand, by definition of x^*, it holds that:

$$f(\ell(A_i))/\ell(A_i) \geq f(x^*)/x^*$$

for all i (indeed, for any load). Moreover, if a PM has been assigned with a load $\ell(d_{i_j})$ bigger than x^*, it also holds that

$$f(\ell(A_i))/\ell(A_i) \geq f(\ell(d_{i_j}))/\ell(d_{i_j}).$$

Hence, we obtain the following inequality:

$$f(\ell(A_i)) \geq \sum_{d_{i_j} \in B_i} f(\ell(d_{i_j})) + \sum_{d_{i_j} \in S_i} \frac{f(x^*)}{x^*} \ell(d_{i_j}).$$

In order to lower bound the power consumption of the solution π^*, we plug the above inequality into the corresponding equation:

$$\begin{aligned} P(\pi^*) &= \sum_{A_i \neq \emptyset} f(\ell(A_i)) \\ &\geq \sum_{A_i \neq \emptyset} \sum_{d_{i_j} \in B_i} f(\ell(d_{i_j})) + \frac{f(x^*)}{x^*} \sum_{A_i \neq \emptyset} \sum_{d_{i_j} \in S_i} \ell(d_{i_j}), \end{aligned}$$

or, equivalently expressed in more compact notation:

$$P(\pi^*) \geq \sum_{d_i : \ell(d_i) \geq x^*} f(\ell(d_i)) + \frac{f(x^*)}{x^*} \sum_{d_i : \ell(d_i) < x^*} \ell(d_i). \tag{7.6}$$

Consider now Algorithm 2. Let us denote by π a solution that Algorithm 2 gives for a particular instance. Also, let us denote by $\hat{A}_i$ the load assigned by Algorithm 2 to PM s_i. Note that due to the design of the algorithm, after the last VM has been assigned, either there is only one loaded PM whose current load is smaller than $x^*/2$, or every loaded PM has a load at least $x^*/2$. We study these two cases separately.

Case 1: $\ell(\hat{A}_i) \geq x^*/2$ for all i. In this case, in a solution provided by π there are PMs with two types of load: those that are loaded with one VM whose load is no smaller than x^*, and those that are loaded with VMs whose load is strictly smaller than x^*, nonetheless, their total load is bigger than $x^*/2$. Note that due to the design of the algorithm, none of the PMs in the second group has a load bigger than x^*. Let us denote by B the set of VMs with load at least x^*, and D_s the set of VMs with load less than x^*. Therefore, it holds:

$$\begin{aligned} P(\pi) &= \sum_{d\in B} f(\ell(d)) + \sum_{\frac{x^*}{2}\le \ell(\hat{A}_i)\le x^*} f(\ell(\hat{A}_i)) \\ &\le \sum_{d\in B} f(\ell(d)) + \frac{f(\frac{x^*}{2})}{\frac{x^*}{2}}\ell(D_s). \end{aligned}$$

Computing the ratio ρ between $P(\pi)$ and $P(\pi^*)$, we obtain the following inequality:

$$\begin{aligned} \rho &\le \frac{\sum_{d\in B} f(\ell(d)) + \frac{f(\frac{x^*}{2})}{\frac{x^*}{2}}\ell(D_s)}{\sum_{d\in B} f(\ell(d)) + \frac{f(x^*)}{x^*}\ell(D_s)} \\ &\le \frac{\frac{f(\frac{x^*}{2})}{\frac{x^*}{2}}\ell(D_s)}{\frac{f(x^*)}{x^*}\ell(D_s)} \\ &= 2\frac{f(\frac{x^*}{2})}{f(x^*)} = 2\left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right). \end{aligned}$$

Case 2: there exists s_i such that $\ell(\hat{A}_i) < x^*/2$. In this case, π gives solutions with three types of loaded PMs: those that are loaded with one VM whose load is bigger than x^*, those that are loaded with VMs whose load is strictly smaller than x^*, but which total load is at least $x^*/2$, and one PM whose total load is is strictly smaller than $x^*/2$. Let us denote such a PM by s'. Therefore, it holds:

$$\begin{aligned} P(\pi) &= \sum_{d\in B} f(\ell(d)) + \sum_{\frac{x^*}{2}\le \ell(\hat{A}_i)\le x^*} f(\ell(\hat{A}_i)) + f(\ell(\hat{A}_{s'})) \\ &\le \sum_{d\in B} f(\ell(d)) + \frac{f(\frac{x^*}{2})}{\frac{x^*}{2}}\Big(\ell(D_s) - \ell(\hat{A}_{s'})\Big) + f(\ell(\hat{A}_{s'})) \\ &= \sum_{d\in B} f(\ell(d)) + \frac{f(\frac{x^*}{2})}{\frac{x^*}{2}}\Big(\ell(D_s) - \ell(\hat{A}_{s'})\Big) + \ell(\hat{A}_{s'})^\alpha + b. \end{aligned}$$

Let us denote the latter expression by $\Pi(\pi)$. Computing the ratio ρ between $P(\pi)$ and $P(\pi^*)$, we obtain the following inequality:

$$\begin{aligned}
\rho &\leq \frac{\Pi(\pi)}{\sum_{d\in B} f(\ell(d)) + \frac{f(x^*)}{x^*}\ell(D_s)} \\
&\leq 2\left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right) + \frac{\ell(\hat{A}_{s'})^\alpha - \ell(\hat{A}_{s'})\frac{f(\frac{x^*}{2})}{\frac{x^*}{2}} + b}{\frac{f(x^*)}{x^*}\ell(D_s)} \\
&\leq 2\left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right) + \frac{\ell(\hat{A}_{s'})^\alpha + b}{\frac{f(x^*)}{x^*}\ell(D_s)} \\
&\leq 2\left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right) + \frac{(\frac{x^*}{2})^\alpha + b}{\frac{f(x^*)}{x^*}\ell(D_s)} \\
&= \left(1-\frac{1}{\alpha}\left(1-\frac{1}{2^\alpha}\right)\right)\left(2+\frac{x^*}{\ell(D_s)}\right).
\end{aligned}$$

Since $x^*/\ell(D_s)$ is always positive, the competitive ratio of Algorithm 2 is equal to $2^{\alpha-1} + x^*/\ell(D_s)$. Observe that, when no VM d has load $\ell(d) < x^*$, i,e., $S = \emptyset$, $P(\pi)$ and $P(\pi^*)$ are equal. Hence, the competitive ratio is 1. ■

Theorem 22. *There exists an online algorithm for $(C,\cdot)$-VMA when $x^* \geq C$ that achieves competitive ratio $\rho \leq \frac{2b}{C}\left(1+\frac{1}{(\alpha-1)2^\alpha}\right)\left(2+\frac{C}{\ell(D)}\right)$.*

Proof: We proceed with the analysis of the competitive ratio of Algorithm 2 in the case when $x^* \geq C$. The analysis uses the same technique used in the proof for the previous theorem. Hence, we just show the difference.

On the one hand, when $x^* \geq C$, it holds that $f(\ell(A_i))/\ell(A_i) \geq f(C)/C$ due to the fact that $f(x)/x$ is monotone decreasing in interval $(0, C]$. It is also obvious that all the PMs will be loaded no more C. As a result, the optimal power consumption for $(C,\cdot)$-VMA can be bounded by

$$P(\pi^*) \geq \frac{f(C)}{C}\ell(D).$$

On the other hand, the solution given by Algorithm 2 can also be upper bounded. We consider the following two cases.

Case 1: $\ell(\hat{A}_i) \geq C/2$ for all i. In this case, every PM will be loaded between $C/2$ and C. Consequently,

$$P(\pi) = \sum_{\frac{C}{2}\leq \ell(\hat{A}_i)\leq C} f(\ell(\hat{A}_i)) \leq \frac{f(\frac{C}{2})}{\frac{C}{2}}\ell(D).$$

The competitive ratio ρ then satisfies

$$\rho \leq \frac{\frac{f(\frac{C}{2})}{\frac{C}{2}}\ell(D)}{\frac{f(C)}{C}\ell(D)} = 2\frac{f(\frac{C}{2})}{f(C)} \leq \frac{2b}{C}\left(1+\frac{1}{(\alpha-1)2^\alpha}\right).$$

Case 2: there exists s_i such that $\ell(\hat{A}_i) < C/2$. In this case, it holds:

$$\begin{aligned}
P(\pi) &= \sum_{\frac{C}{2} \leq \ell(\hat{A}_i) \leq C} f(\ell(\hat{A}_i)) + f(\ell(\hat{A}_{s'})) \\
&\leq \frac{f(\frac{C}{2})}{\frac{C}{2}} \Big(\sum_{d_i : \ell(d_i) \leq C} \ell(d_i) - \ell(\hat{A}_{s'}) \Big) + f(\ell(\hat{A}_{s'})) \\
&= \frac{f(\frac{C}{2})}{\frac{C}{2}} \Big(\ell(D) - \ell(\hat{A}_{s'}) \Big) + \ell(\hat{A}_{s'})^\alpha + b.
\end{aligned}$$

The competitive ratio ρ then satisfies

$$\begin{aligned}
\rho &\leq \frac{P(\pi)}{\frac{f(C)}{C}\ell(D)} \leq \frac{2b}{C}\left(1 + \frac{1}{(\alpha-1)2^\alpha}\right) + \\
&+ \frac{\ell(\hat{A}_{s'})^\alpha - \ell(\hat{A}_{s'})\frac{f(\frac{C}{2})}{\frac{C}{2}} + b}{\frac{f(C)}{C}\ell(D)} \\
&\leq \frac{2b}{C}\left(1 + \frac{1}{(\alpha-1)2^\alpha}\right) + \frac{\ell(\hat{A}_{s'})^\alpha + b}{\frac{f(C)}{C}\ell(D)} \\
&\leq \frac{2b}{C}\left(1 + \frac{1}{(\alpha-1)2^\alpha}\right) + \frac{(\frac{C}{2})^\alpha + b}{\frac{f(C)}{C}\ell(D)} \\
&= \frac{2b}{C}\left(1 + \frac{1}{(\alpha-1)2^\alpha}\right)\left(2 + \frac{C}{\ell(D)}\right).
\end{aligned}$$

■

7.4.2.2 Approximation Ratio for Algorithm *VMA2*

We prove the approximation ratio of Algorithm 3 in the following theorem.

Theorem 23. *For a system with unbounded number of PMs, Algorithm 3 achieves a competitive ratio of* 1 *if no VM* d_i *has* $\ell(d_i) < x^*$*, and of* $2^{\alpha-1} + x^* / \sum_{d_i:\ell(d_i)<x^*} \ell(d_i)$*, otherwise.*

Proof: Maintaining the same notation we used for proving Theorem 21, let us denote by π a solution that Algorithm 3 gives for a particular instance and by $\hat{A}_i$ the load assigned by Algorithm 3 to PM s_i. Note that due to the design of the algorithm, after the last VM has been assigned, either there is only one loaded PM whose current load is smaller than x^*, or every loaded PM has a load bigger than x^*. We study these two cases separately.

Case 1: $\ell(\hat{A}_i) \geq x^*$ for all i. In this case, in a solution provided by π there are PMs with two types of load: those that are loaded with one VM whose load is no smaller than x^*, and those that are loaded with VMs whose load is strictly smaller than x^*, nonetheless, their total load is bigger than x^*. Note that due to the design of the algorithm, none of the PMs in the second group has a load bigger than $2x^*$. Let us denote by B the set of demands with load at least x^*, and S the set of demands with load less than x^*. Therefore, it holds:

$$\begin{aligned}
P(\pi) &= \sum_{d \in B} f(\ell(d)) + \sum_{x^* \leq \ell(\hat{A}_i) \leq 2x^*} f(\ell(\hat{A}_i)) \\
&\leq \sum_{d \in B} f(\ell(d)) + \frac{f(2x^*)}{2x^*} \sum_{\ell(d \in S} \ell(d).
\end{aligned}$$

Computing the ratio ρ between $P(\pi)$ and the expression for $P(\pi^*)$ from Eq. (7.6), we obtain the following inequality:

$$\rho \leq \frac{\sum_{d \in B} f(\ell(d)) + \frac{f(2x^*)}{2x^*} \sum_{d \in S} \ell(d)}{\sum_{d \in B} f(\ell(d)) + \frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \leq \frac{\frac{f(2x^*)}{2x^*} \sum_{d \in S} \ell(d)}{\frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \leq 2^{\alpha-1}. \quad (7.7)$$

Case 2: there exists s_i such that $\ell(\hat{A}_i) < x^*$. In this case, π gives solutions with three types of loaded PMs: those that are loaded with one VM whose load is bigger than x^*, those that are loaded with VMs whose load is strictly smaller than x^*, but which total load is bigger than x^*, and one PM whose total load is is strictly smaller than x^*. Let us denote such a PM by s'. Therefore, it holds:

$$\begin{aligned}
P(\pi) &= \sum_{d \in B} f(\ell(d)) + \sum_{x^* \leq \ell(\hat{A}_i) \leq 2x^*} f(\ell(\hat{A}_i)) + f(\ell(\hat{A}_{s'})) \\
&\leq \sum_{d \in B} f(\ell(d)) + \frac{f(2x^*)}{2x^*} \Big(\sum_{d \in S} \ell(d) - \hat{A}_{s'} \Big) + f(\ell(\hat{A}_{s'})) \\
&= \sum_{d \in B} f(\ell(d)) + \frac{f(2x^*)}{2x^*} \Big(\sum_{d \in S} \ell(d) - \hat{A}_{s'} \Big) + \ell(\hat{A}_{s'})^\alpha + b.
\end{aligned}$$

Let us denote the latter expression by $\Pi(\pi)$. Computing the ratio ρ between $P(\pi)$ and $P(\pi^*)$, we obtain the following inequality:

$$\begin{aligned}
\rho &\leq \frac{\Pi(\pi)}{\sum_{d \in B} f(\ell(d)) + \frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \quad (7.8) \\
&\leq 2^{\alpha-1} + \frac{\ell(\hat{A}_{s'})^\alpha - \hat{A}_{s'} \frac{f(2x^*)}{2x^*} + b}{\frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \\
&\leq 2^{\alpha-1} + \frac{\ell(\hat{A}_{s'})^\alpha + b}{\frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \\
&\leq 2^{\alpha-1} + \frac{(x^*)^\alpha + b}{\frac{f(x^*)}{x^*} \sum_{d \in S} \ell(d)} \\
&= 2^{\alpha-1} + \frac{x^*}{\sum_{d \in S} \ell(d)}.
\end{aligned}$$

Since $x^*/\sum_{d\in S}\ell(d)$ is always positive, the competitive ratio of Algorithm 2 is equal to $2^{\alpha-1}+x^*/\sum_{d\in S}\ell(d)$. Observe that, when no VM d has load $\ell(d)<x^*$, i,e., $S=\emptyset$, equations (7.7) and (7.8) become $\frac{P(\pi)}{P(\pi^*)}\leq 1$. ■

7.4.2.3 Upper Bounds for $(\cdot, 2)$-VMA problem

We now present an algorithm (detailed in Algorithm 4) for $(\cdot, 2)$-VMA problem and show an upper bound on its competitive ratio. A_1 and A_2 are the sets of VMs assigned to PMs s_1 and s_2, respectively, at any given time.

Algorithm 4: Online algorithm for $(\cdot, 2)$-VMA.

```
for each VM $d_i$ do
    if $\ell(d_i)+\ell(A_1)\leq (b/(2^\alpha-2))^{1/\alpha}$ or $\ell(A_1)\leq\ell(A_2)$ then
        $d_i$ is assigned to $s_1$;
    else
        $d_i$ is assigned to $s_2$;
```

We prove the approximation ratio of Algorithm 4 in the following theorem.

Theorem 24. *There exists an online algorithm for $(\cdot, 2)$-VMA that achieves the following competitive ratios.*

$$\begin{aligned} \rho &= 1, && \text{for } \ell(D)\leq\left(\frac{b}{2^\alpha-2}\right)^{1/\alpha},\\ \rho &\leq \max\left\{2,\left(\frac{3}{2}\right)^{\alpha-1}\right\}, && \text{for } \ell(D)>\left(\frac{b}{2^\alpha-2}\right)^{1/\alpha}. \end{aligned}$$

Proof: Consider Algorithm 4 shown above. If $\ell(D)\leq(b/(2^\alpha-2))^{1/\alpha}$, then the competitive ratio is 1 as we show. Algorithm 4 assigns all the VMs to PM s_1. On the other hand, the optimal offline algorithm also assigns all the VMs to one PM. To prove it, it is enough to show that $\ell(D)^\alpha+b<\ell(A_1)^\alpha+\ell(A_2)^\alpha+2b$. Using that $\ell(A_1)^\alpha+\ell(A_2)^\alpha>2\,(\ell(D)/2)^\alpha$ and manipulating, it is enough to prove $\ell(D)<2\,(b/(2^\alpha-2))^{1/\alpha}$. This is true for $\ell(D)\leq(b/(2^\alpha-2))^{1/\alpha}$.

We consider now the case $(b/(2^\alpha-2))^{1/\alpha}<\ell(D)<2\,(b/(2^\alpha-2))^{1/\alpha}$. Within this range, for the optimal algorithm is still better to assign all VMs to one PM, as shown. Then, the competitive ratio ρ is

$$\rho=\frac{\ell(A_1)^\alpha+\ell(A_2)^\alpha+2b}{\ell(D)^\alpha+b}\leq\frac{\ell(D)^\alpha+2b}{\ell(D)^\alpha+b}<2. \tag{7.9}$$

Consider any given step after $\ell(D)\geq 2\,(b/(2^\alpha-2))^{1/\alpha}$. Within this range, the optimal algorithm may assign the VMs to one or both PMs. If the optimal algorithm assigns to one PM, Inequality 7.9 applies. Otherwise, the competitive ratio ρ is

$$\begin{aligned}\rho &= \frac{\ell(A_1)^\alpha + \ell(A_2)^\alpha + 2b}{2(\ell(D)/2)^\alpha + 2b} \leq 2^{\alpha-1}\frac{\ell(A_1)^\alpha + \ell(A_2)^\alpha}{\ell(D)^\alpha} \\ &= 2^{\alpha-1}\frac{\ell(A_1)^\alpha/\ell(A_2)^\alpha + 1}{(\ell(A_1)/\ell(A_2)+1)^\alpha}.\end{aligned}$$

Then, in order to obtain a ratio at most $x^\alpha/2$, where x will be set later, it is enough to guarantee

$$\begin{aligned}2^{\alpha-1}\frac{\ell(A_1)^\alpha/\ell(A_2)^\alpha + 1}{(\ell(A_1)/\ell(A_2)+1)^\alpha} &\leq \frac{x^\alpha}{2} \\ \frac{(\ell(A_1)/\ell(A_2))^\alpha + 1}{(\ell(A_1)/\ell(A_2)+1)^\alpha} &\leq \left(\frac{x}{2}\right)^\alpha.\end{aligned}$$

Without loss of generality, assume $\ell(A_1) \leq \ell(A_2)$. This implies that $(\ell(A_1)/\ell(A_2))^\alpha \leq \ell(A_1)/\ell(A_2)$. Then, it is enough to have

$$\frac{\ell(A_1)/\ell(A_2) + 1}{(\ell(A_1)/\ell(A_2)+1)^\alpha} \leq \left(\frac{x}{2}\right)^\alpha.$$

Let us now define $\ell(A_1) + \ell = \ell(A_2)$ for some $\ell \geq 0$. Manipulating and replacing, it is enough to show

$$\frac{\ell}{\ell(A_1)} \leq \frac{2 - (2/x)^{\alpha/(\alpha-1)}}{(2/x)^{\alpha/(\alpha-1)} - 1}. \tag{7.10}$$

If Inequality 7.10 holds the theorem is proved. Otherwise, the following claim is needed.

Claim 1. *If $\ell(D) \geq 2\,(b/(2^\alpha - 2))^{1/\alpha}$, then there must exist a VM d_i in D such that $\ell(d_i) \geq |\ell(A_2) - \ell(A_1)|$.*

Proof: If $\ell(A_2) = \ell(A_1)$ the claim follows trivially. Assume that $\ell(A_2) \neq \ell(A_1)$. Consider any given time when $\ell(D) \geq 2\,(b/(2^\alpha - 2))^{1/\alpha}$. For the sake of contradiction, assume that for all $d_i \in D$ it is $\ell(d_i) < |\ell(A_2) - \ell(A_1)|$. Let $d_1, d_2, \ldots, d_r$ be the order in which the VMs were revealed to Algorithm 4. And let the respective sets of VMs be called $D_i = \{d_j | j \in [1, i]\}$, that is $D_r = D$. Given that $\ell(D) \geq 2\,(b/(2^\alpha - 2))^{1/\alpha} > (b/(2^\alpha - 2))^{1/\alpha}$, the VM d_r was assigned to the PM with smaller load. Then, either $\ell(d_r) \geq |\ell(A_2) - \ell(A_1)|$ which would be a contradiction, or if $\ell(d_r) < |\ell(A_2) - \ell(A_1)|$ the PM with the smaller load before and after assigning d_r is the same. The argument can be repeated iteratively backwards for each d_{r-1}, d_{r-2}, etc. until, for some $j \in [1, r)$, either it is $\ell(d_j) \geq |\ell(A_2) - \ell(A_1)|$ reaching a contradiction, or the total load is $\ell(D_j) < (b/(2^\alpha - 2))^{1/\alpha}$. If the latter is the case, we know that for $i \in [1, j]$ every d_i was assigned to s_1. Recall that for $i \in (j, r]$ each d_i was assigned to the same PM. And, given that d_{j+1} is the first VM for which the total load is at least $(b/(2^\alpha - 2))^{1/\alpha}$, that PM is s_2. But then, we have $\ell(A_2) < \ell(A_1) < (b/(2^\alpha - 2))^{1/\alpha}$, which is a contradiction with the assumption that $\ell(D) \geq 2\,(b/(2^\alpha - 2))^{1/\alpha}$. ■

Using Claim 1 we know that there exists a d_i in the input such that

$$\ell(d_i) \geq \ell > \ell(A_1)\frac{2-(2/x)^{\alpha/(\alpha-1)}}{(2/x)^{\alpha/(\alpha-1)}-1}.$$

From the latter, it can be seen that if $x \geq 2(3/4)^{\frac{\alpha-1}{\alpha}}$, then we have that $\ell > 2\ell(A_1)$. Then, the competitive ratio ρ is

$$\begin{aligned}\rho &= \frac{\ell(A_1)^\alpha + (\ell(A_1)+\ell)^\alpha + 2b}{(2\ell(A_1))^\alpha + \ell^\alpha + 2b} \\ &\leq \frac{\ell(A_1)^\alpha + (\ell(A_1)+\ell)^\alpha}{(2\ell(A_1))^\alpha + \ell^\alpha}.\end{aligned}$$

Using calculus, this ratio is maximized for $\ell = 2\ell(A_1)$ for $\ell \geq 2\ell(A_1)$. Then, we have $\rho \leq (1+3^\alpha)/(2\cdot 2^\alpha)$. Then, in order to obtain a ratio at most $x^\alpha/2$, it is enough to guarantee $(1+3^\alpha)/(2\cdot 2^\alpha) \leq x^\alpha/2$ which yields $x \geq ((1+3^\alpha)/2^\alpha)^{1/\alpha}$.

Given that, for any $\alpha \geq 1$, it holds:

$$2(3/4)^{1-1/\alpha} \geq ((1+3^\alpha)/2^\alpha)^{1/\alpha}.$$

Then, the competitive ratio is $\rho \leq (2(3/4)^{1-1/\alpha})^\alpha/2 = (3/2)^{\alpha-1}$. ■

7.5 Experimental Evaluation

In this section we experimentally evaluate the performance of two of the online algorithms proposed in Section 7.4.2, namely *VMA1* and *VMA2*, and compare them to other state-of-the-art online placement algorithms.

Both *VMA1* and *VMA2* have been extended to handle a bounded number of PMs when necessary. This is achieved by assigning the incoming task to the least loaded machine when no more new PMs are available. Similarly, *VMA2* has also been extended to deal with the bounded capacity case, when required, by using a threshold of $\min\{x^*, C\}$ and checking whether a new load fits into the targeted PM. In case a new load does not fit, it is assigned to the first possible PM with available resources. Although the competitive ratios of *VMA2* are worse than the ones from *VMA1*, it exhibits a better behavior in simulation.

7.5.1 Experimental Setup

The performance of both *VMA1* and *VMA2* is first compared with a lower bound, denoted *LBVMA*, that is obtained as follows. The input VMs are sorted in non-increasing order of their loads. Then, using this order, as many VMs as possible with load at least x^* are assigned to different PMs. Let L be total load of the VMs still unassigned. If there are $\lfloor L/x^* \rfloor$ still unused

PMs they will be used. Otherwise all PMs will be used. Finally, the load L is assigned among all used PMs as if it could be infinitely divided (i.e., as a fluid), using a water-filling algorithm [38].

We evaluate both $(\cdot, m)$-VMA and (C, m)-VMA problems, therefore, in the (C, m)-VMA case a VM can only be assigned to a PM if the latter has sufficient capacity to host it. We test both algorithms *VMA1* and *VMA2* and also compare them with the following algorithms proposed in the literature:

- Random Fit (RF) [116]: It chooses a PM for each VM uniformly at random among the PM. If the chosen PM cannot allocate the load of the VM, the process is repeated, until the VM is assigned to a PM.

- Next Fit (NF) [116]: Starting initially at the first PM, each new VM is assigned to the next PM after the latest PM to which a VM was assigned (in a cyclic fashion) and with sufficient capacity to host it.

- Least Full First (LFF) [116]: Each new VM is assigned to (one of) the least loaded PM(s) in the system with enough capacity to host it.

- Striping (S) [96]: Each new VM is assigned to (one of) the PM(s) with the smallest number of VMs assigned and with enough capacity to host it.

- Watts per Core (WC) [96]: Assigns each new VM to the PM whose power would suffer the smallest increase and with enough capacity to host it.

- First Fit (FF) [116]: Starting initially at the first PM, each new VM is placed in the first PM that can host it.

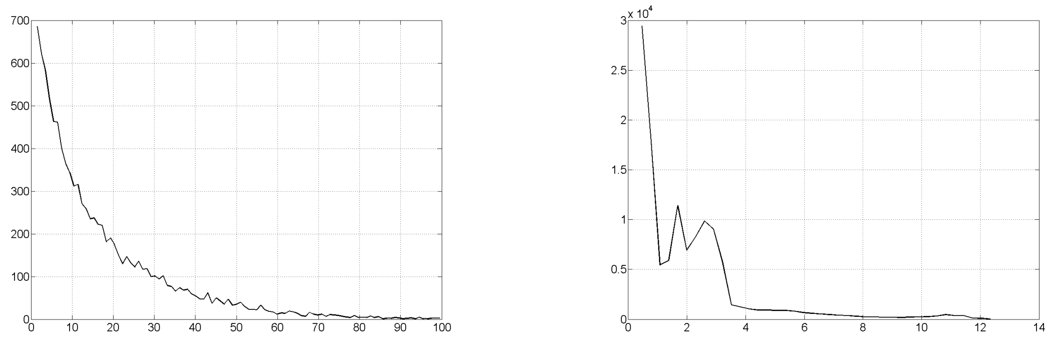
- Round Robin (RR) [96]: Like FF but, after the first VM is assigned, the search starts from the latest PM in which a VM was allocated.

- Packing (P) [96]: Each new VM is assigned to (one of) the PM(s) with the largest number of VMs assigned provided that the PM can host it.

- Most Full First (MFF) [116]: Each new VM is assigned to the most loaded PM in which it fits.

Observe that, given its nature, First Fit, Round Robin, Packing and Most Loaded First can be only considered for the (C, m)-VMA problem. If PMs had infinite capacity, these algorithms would place all VMs in only one PM. The remaining algorithms are evaluated for both $(\cdot, m)$-VMA and (C, m)-VMA.

The behavior of the aforementioned algorithms is evaluated by inputting the two sets of traces, synthetic and real, shown in Figure 7.1. We call them *Trace A* and *Trace B*, respectively. Trace A is generated by randomly choosing the load of each VM following a power-law distribution with exponential cutoff, which has been chosen so 100% is the maximum task load of a VM.



(a) Trace A (synthetic traces) (b) Trace B (Google traces)

Figure 7.1: VM load distributions used in the evaluations.

We randomly select 10000 integer loads using this distribution. This leads us to the VM load distribution shown in Figure 7.1(a).

Trace B is obtained from public Google traces [88]. We extract all the tasks from these traces, assuming that each task is an independent VM. The VMs (tasks) are sorted by the time at which they join the system. The task load of a VM is the maximum CPU load of the task. The trace then contains 124885 VMs with loads varying between 0.31% and 12.5%. The resulting VM load distribution can be seen in Figure 7.1(b). The load values of the VMs for both distributions are given in percentage in order to scale them up depending on the maximum capacity of a PM in the (C, m)-VMA case or to an appropriate value in the $(\cdot, m)$-VMA case.

Each execution of the algorithms is run with a fixed number of PMs. This number of PMs increases from 1 to the number of VMs in the trace being used. This allows us to see how the power consumption and how the algorithms behavior evolve when the number of available PMs in the system varies. Finally, in order to evaluate both $(\cdot, m)$-VMA and (C, m)-VMA, we *emulate* different PMs by determining their α, b, μ and x^* parameters as well as the PM maximum capacity or the maximum task load when it corresponds. Then we run the proposed algorithms for each one of these emulated PMs and compare the influence of the different values for these parameters on the final results.

7.5.2 Experimental Results for $(\cdot, m)$-VMA

We first evaluate $(\cdot, m)$-VMA. The first step is to define the set of PMs that we are going to use to evaluate it. To do so, we fix the values of α, b and x^* and compute μ depending on the previous parameters. In particular, we used $\alpha = \{1.5, 2, 2.5, 3\}$ and $x^* = \{10, 30, 50, 75, 100, 130, 150, 300, 500, 750, 1000\}$ (given in (Giga)*Cycles per Second* (GCPS) following the conclusions from Chapter 6). The values of b are determined by interpolation of the baseline costs of `Nemesis`, `Survivor` and `Erdos`, whose values for b ($\sim$ 85 W, 67 W and 215 W) are known from the experiments performed in Chapter 6. As we mentioned, the values

Table 7.3: Simulation parameters for a set of machines for the $(\cdot, m)$-VMA case.

$(\cdot,\cdot)$-VMA case					
b	x^* [GCPS]	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 2.5$	$\alpha = 3$
73.05	10	1.46E-13	7.31E-19	4.87E-24	3.65E-29
92.15	30	3.55E-14	1.02E-19	3.94E-25	1.71E-30
111.25	50	1.99E-14	4.45E-20	1.33E-25	4.45E-31
135.125	75	1.32E-14	2.40E-20	5.85E-26	1.60E-31
159	100	1.01E-14	1.59E-20	3.35E-26	7.95E-32
187.65	130	8.01E-15	1.11E-20	2.05E-26	4.27E-32
206.75	150	7.12E-15	9.19E-21	1.58E-26	3.06E-32
350	300	4.26E-15	3.89E-21	4.73E-27	6.48E-33
541	500	3.06E-15	2.16E-21	2.04E-27	2.16E-33
779.75	750	2.40E-15	1.39E-21	1.07E-27	9.24E-34
1018.5	1000	2.04E-15	1.02E-21	6.79E-28	5.09E-34

of b, α and x^* determine the value of μ. These combination of parameters results in 44 different instances of PMs which are shown in Table 7.3.

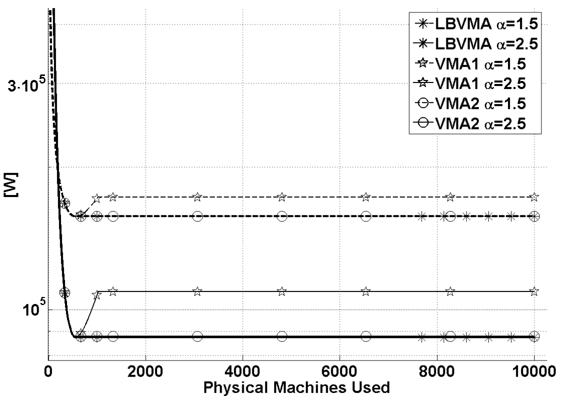
Additionally, taking advantage of the fact that the task loads from Trace A and B are given in percentage, and in order to study the importance of the x^* to task load ratio, we define the maximum VM load λ as the maximum load that a VM arriving to the system can have. Therefore, the load of the VMs arriving to the system will be the product of the task load (in percentage) and λ. λ will take the following values: 10, 30 and 100 GCPS.

We study three different scenarios. In the first one we study the effect of α for different values of λ and x^* when using *VMA1*, *VMA2*, and the lower bound *LBVMA*. Second and third scenarios are devoted to compare our proposed algorithms with the state-of-the-art algorithms, always keeping *LBVMA* as a reference. In the second scenario we study the relevance of λ while keeping α and x^* constant. Finally, in the third one, we study the effect of having different values of x^* while λ and α remain unaltered.

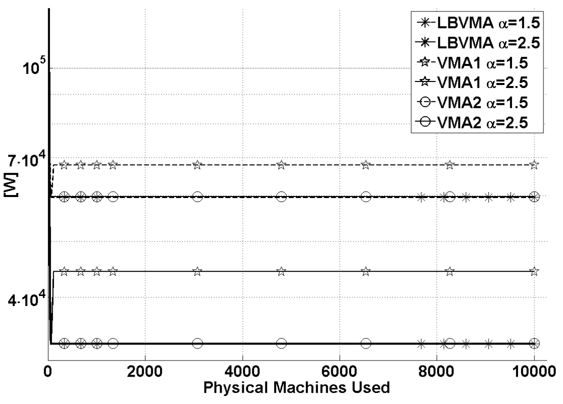
Scenario 1 compares the power consumed by partitions obtained with *VMA1* or *VMA2* and for Trace A and Trace B. We compare these results to the ones achieved by *LBVMA*, that lower bounds the optimal power consumption. The results obtained are presented as graphs in which the power consumed is represented as a function of the number of PMs used.

Figure 7.2 and Figure 7.3 show the results for Trace A and Trace B, for 2 different values of x^*, 30 and 300 GCPS, and 2 different values of λ, 10 and 100 GCPS. We can clearly see how the power consumption is smaller for larger values of α once the optimal number of used PMs is reached, mainly conditioned by how μ decreases as α increases (See Table 7.3. Also, as it can be observed, there is no qualitative difference in the solutions when α varies for a given configuration (Similar results are obtained with other values of x^* and maximum task load).

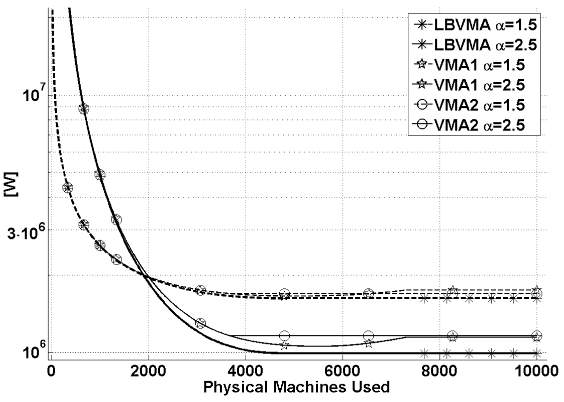
Regarding the performance of the algorithms, we can see how the power consumed by the partitions found with *VMA2* is lower, in all cases, than the ones obtained by *VMA1* and is always



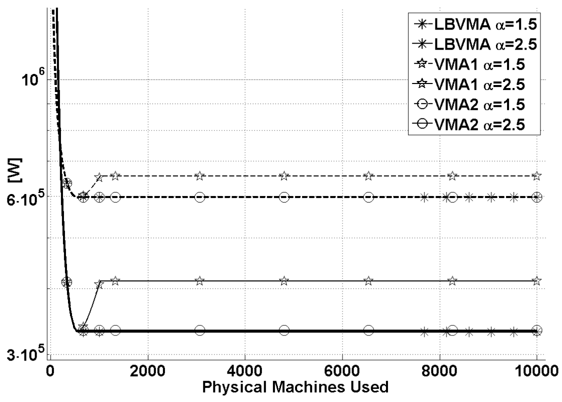
(a) 10 GCPS, $x^* = 30$ GCPS



(b) 10 GCPS, $x^* = 300$ GCPS



(c) 100 GCPS, $x^* = 30$ GCPS

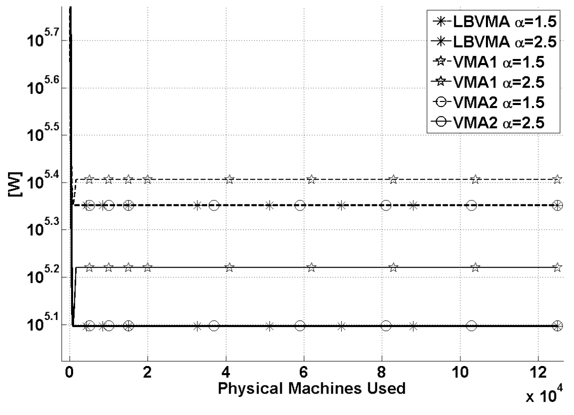


(d) 100 GCPS, $x^* = 300$ GCPS

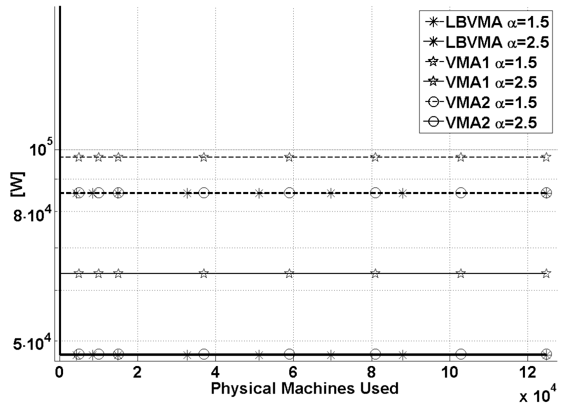
Figure 7.2: $(\cdot, m)$-VMA: Comparing the power consumed by *VMA1* and *VMA2* with the lower bound *LBVMA* for $x^* = \{30, 300\}$ GCPS, $\alpha = \{1.5, 2.5\}$ and $\lambda = \{10, 100\}$ GCPS for Trace A (Synthetic traces).

closer to the lower bound obtained by *LBVMA*. This shows that the performance of *VMA2* is close to the optimal for $(\cdot, m)$-VMA. We can see how, in general, *VMA1* is able to match the results of *VMA2* when the number of PMs is relatively low. However, due to the threshold imposed on the load of the PMs for each algorithm, *VMA2* is able to pack the load in less PMs. We only find an exception in Figure 7.2(c), when $x^*/\lambda < 1/3$ and we are using synthetic traces. In this case *VMA2* exhibits a behavior relatively similar to *VMA1*, not being able to hold to its best power consumption and reducing the quality of the solution when the number of PMs increases. However, this flaw is not replicated when using Trace B, due to the smaller amount of big loads in comparison with Trace B.

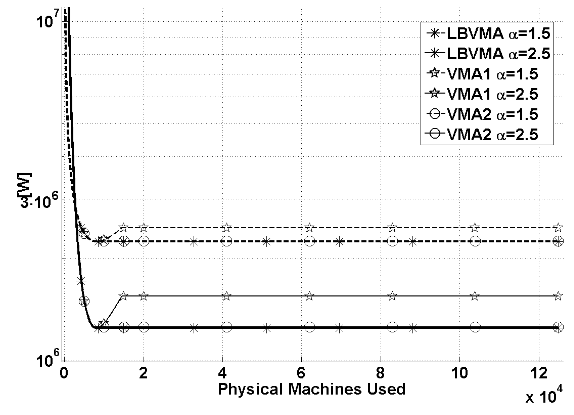
Scenario 2 compares the performance of *LBVMA*, *VMA1* and *VMA2* with the other assignment algorithms proposed in the literature. Here, the values of x^* and α are fixed to 30 GCPS and 2, respectively, while the value of λ varies. In particular we use $\lambda = \{10, 30, 100\}$ GCPS. Figures



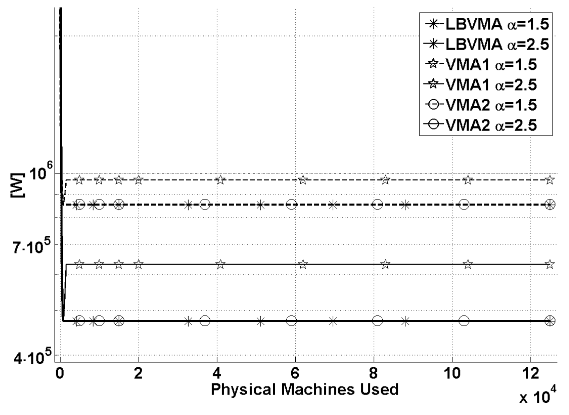
(a) 10 GCPS, $x^* = 30$ GCPS



(b) 10 GCPS, $x^* = 300$ GCPS



(c) 100 GCPS, $x^* = 30$ GCPS



(d) 100 GCPS, $x^* = 300$ GCPS

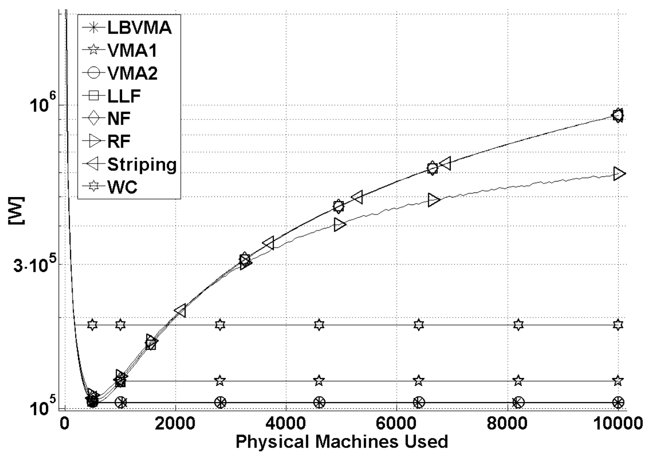
Figure 7.3: $(\cdot, m)$-VMA: Comparing the power consumed by *VMA1* and *VMA2* with the lower bound $LBVMA$ for $x^* = \{30, 300\}$ GCPS, $\alpha = \{1.5, 2.5\}$ and $\lambda = \{10, 100\}$ GCPS for Trace B (Google traces).

7.4 and 7.5 present the results for Trace A and Trace B[3].

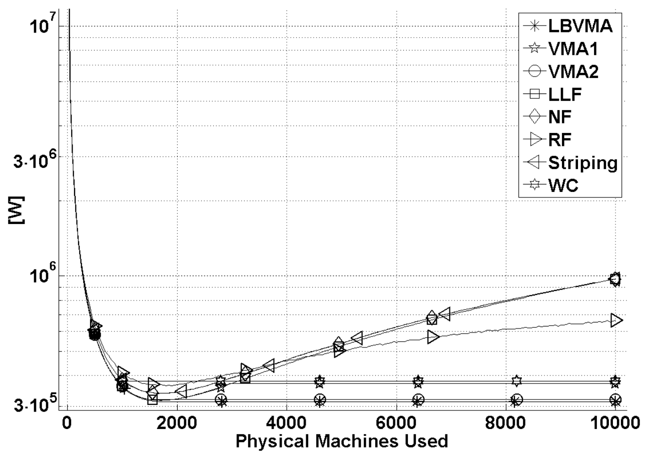
We can easily see, in general, 3 different trends in Figures 7.4 and 7.5. The first trend would include *LBVMA*, *VMA1* and *VMA2*; then we have a second one including *Striping*, *RF*, *NF* and *LLF*; and, finally, in some sort of no-man's land, we have *WC*. These trends have their origin in power awareness. While *LBVMA*, *VMA1*, *VMA2* and *WC* are power aware, the rest are not. According to Figures 7.4 and 7.5, power aware algorithms outperform the non power aware ones.

It is interesting to see how *WC* reduces its power consumption (for Trace A) as the ratio x^*/λ decreases and even performs better than *VMA1* for $\lambda = 100$ GCPS. This does not happen, though, for Trace B due to the smaller average task load, that gives advantage to *VMA1* and *VMA2*. With Trace B, due to its nature, *WC* obtains partitions that use less PMs and hence, because of the

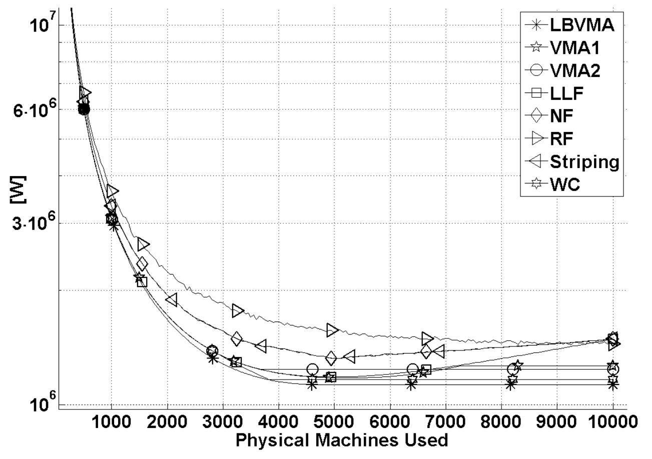
[3]For the sake of clarity, we do not show the power consumption resulting of using only one (or a few) machines and center the figure into more relevant cases



(a) $\lambda = 10$ GCPS

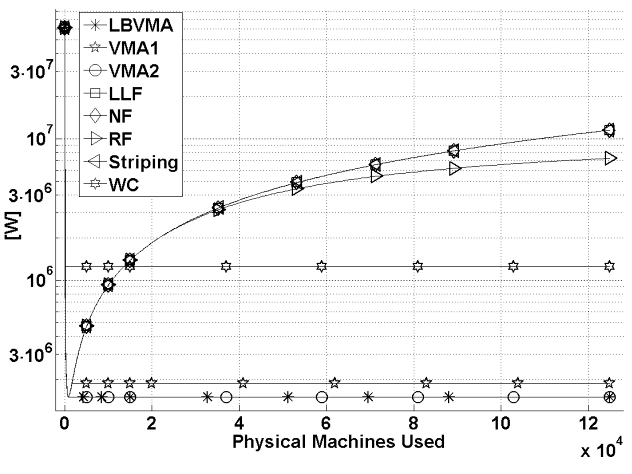


(b) $\lambda = 30$ GCPS

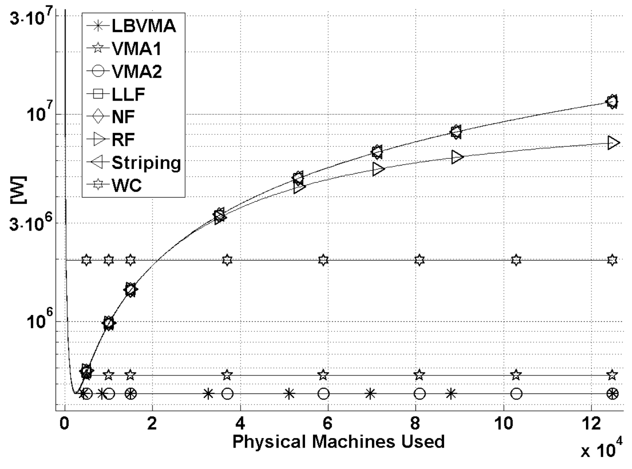


(c) $\lambda = 100$ GCPS

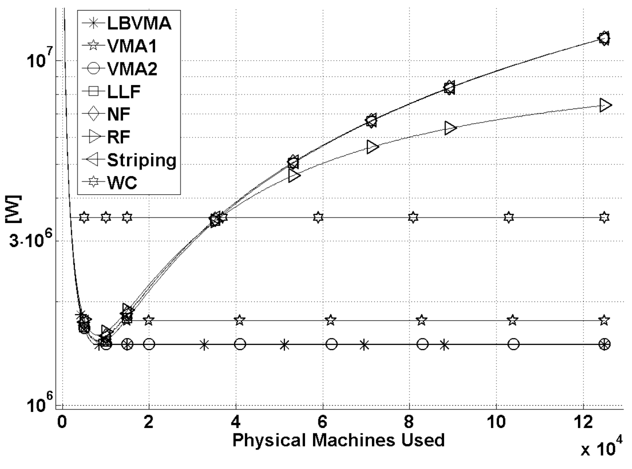
Figure 7.4: $(\cdot, m)$-VMA: Comparing the power consumed by the different assignment algorithms for $x^* = 30$ GCPS, $\alpha = 2$ and $\lambda = \{10, 30, 100\}$ GCPS for Trace A (Synthetic traces).



(a) $\lambda = 10$ GCPS



(b) $\lambda = 30$ GCPS



(c) $\lambda = 100$ GCPS

Figure 7.5: $(\cdot, m)$-VMA: Comparing the power consumed by the different assignment algorithms for $x^* = 30$ GCPS, $\alpha = 2$ and $\lambda = \{10, 30, 100\}$ GCPS for Trace B (Google traces).

superlinear dependence of the power consumption on the load, have higher power consumptions.

Similarly, we can observe that the results in Figure 7.4(c) are tighter. This is a consequence, again, of the low value of x^*/λ, resulting in many PMs not allocating more than 1 or 2 VMs. In fact, steady state for *VMA1*, *VMA2*, *WC* and even *LBVMA* is reached between the 4000 and the 5000 PMs while in the previous cases was reached before the 2000 PMs. Again, this behavior is not replicated with Trace B due to its lower average task load.

Note also that the non power aware algorithms pay a higher power bill due to the use of a larger amount of PMs (in general) with a smaller amount of load per PM, resulting in a very inefficient usage of the available resources. This behavior is consistent for both Traces A and B.

Finally, observe how the larger the ratio x^*/λ, the larger the gap between our proposed algorithms, *VMA1* and *VMA2*, and the other ones. This would be the case when we have tasks that consume a very small amount of CPU in the system.

Let us now analyze the results of the last scenario. Here we keep $\lambda = 30$ GCPS and $\alpha = 2$ constant while we vary the value of x^*. These results are shown in Figure 7.6 for Trace A and in Figure 7.7 for Trace B.

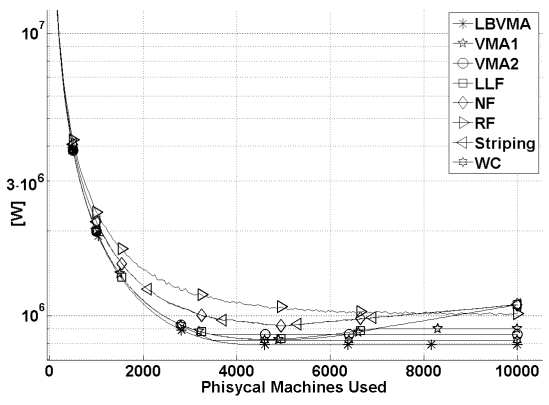
The results are similar for both traces. We can see how for the smallest value, $x^* = 10$ GCPS ($x^*/\lambda = 1/3$) all algorithms achieve a similar result. As the ratio x^*/λ increases, the results obtained by *VMA1* and *VMA2* become better than the ones achieved by *WC*, *LLF*, *NF*, *Striping*, and *RF*, that increase with x^* and, therefore, lead to a larger power consumption. These results are in line with the results from Scenario 1 and are is motivated by the fact that the state-of-the-art algorithms tend to use a large amount of PMs keeping its average load low and, hence, paying a high price because of the b parameter. This, however, is not the case of *WC*, which, on the other hand, obtains a more packed partition, loading PMs beyond x^* and paying an extra cost due to the superlinearity of the power consumption with respect to the load.

7.5.3 Experimental Results for (C, m)-VMA

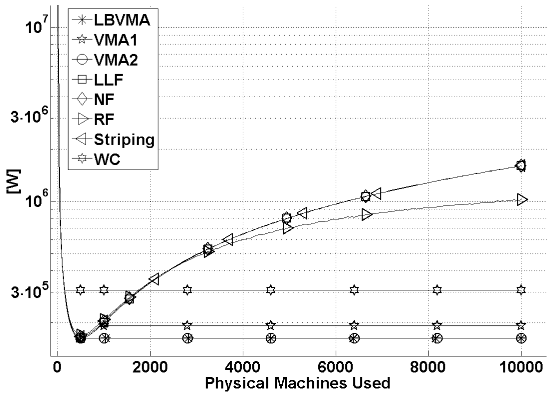
As we did with $(\cdot, m)$-VMA in Subsection 7.5.2, we start by defining the set of PMs we are going to work with. While for $(\cdot, m)$-VMA we assumed that PMs had infinite capacity, in (C, m)-VMA the PMs capacity is bounded. We denote the capacity as C. We define 3 sets of instances that we name after 3 real PMs from our laboratory, `Nemesis`, `Euler`, and `Erdos`. We use, as a reference, their maximum capacity, 11.2, 41.6, and 153.6 GCPS; and approximated idle cost b, 80, 100, and 200 Watts.

Jointly with C and b, we need a value of α and x^* to compute each value of μ. We will use $\alpha = \{1.5, 2, 2.5, 3\}$, and $x^* = \{0.5, 0.65, 0.75, 0.9, 1, 1.1\} \cdot C$. We can now compute the value of μ for each combination of these 4 parameters fully defining, then, our set of PMs. The combination of values for each one of these PM instances can be found in Tables 7.5, 7.6 and 7.4. We base the performance analysis of our proposed algorithms and the other state of the art assignment algorithms on these sets of PM instances.

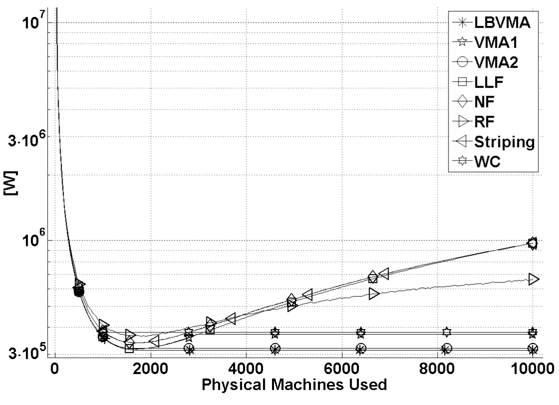
Like for the $(\cdot, m)$-VMA case, we also consider three different scenarios. In the first one we



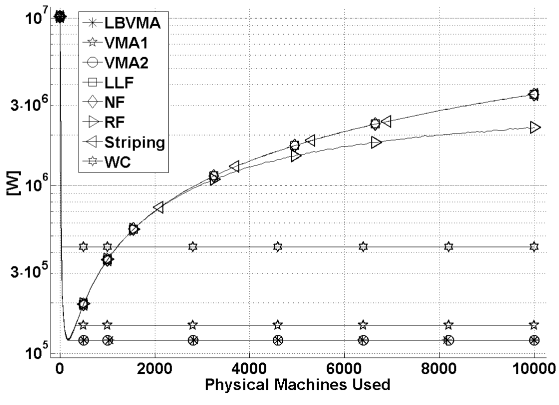
(a) $x^* = 10$ GCPS



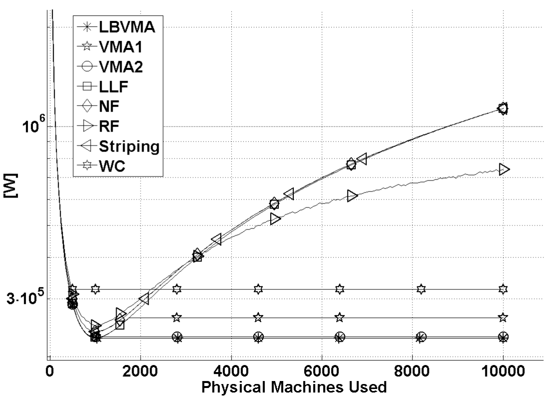
(b) $x^* = 100$ GCPS



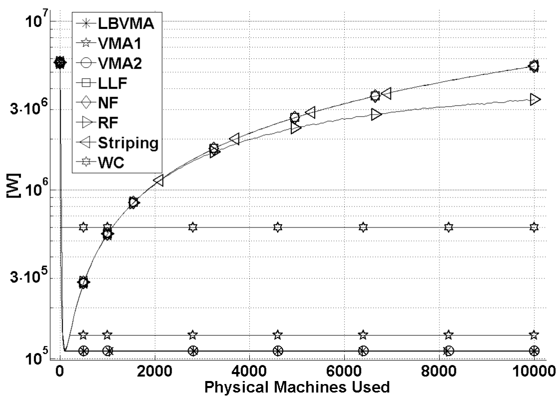
(c) $x^* = 30$ GCPS



(d) $x^* = 300$ GCPS

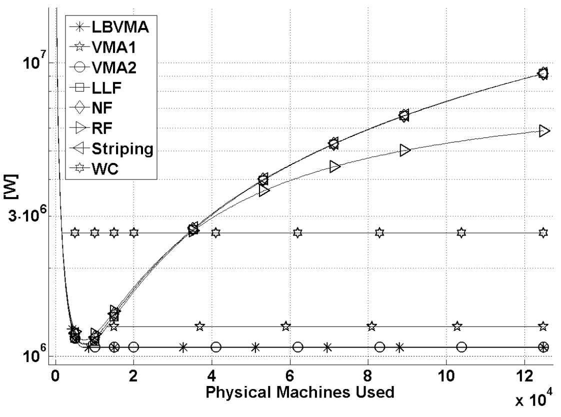


(e) $x^* = 50$ GCPS

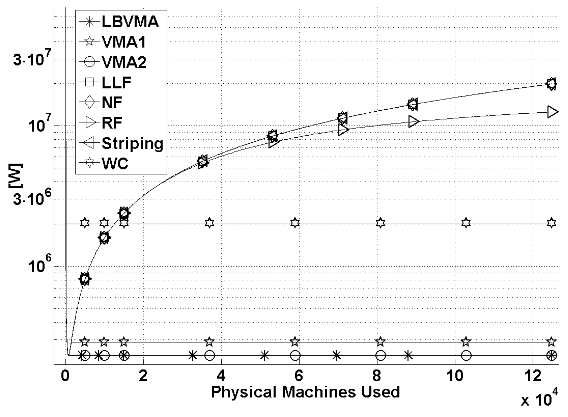


(f) $x^* = 500$ GCPS

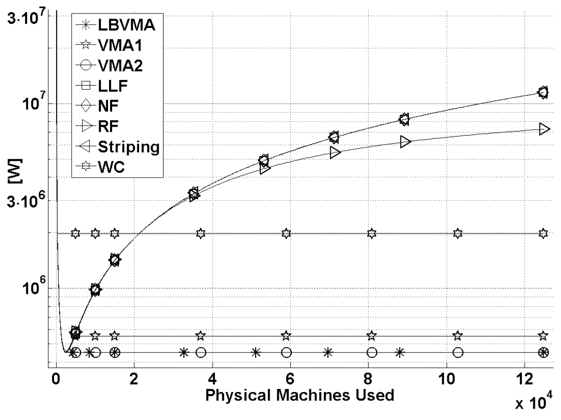
Figure 7.6: $(\cdot, m)$-VMA: Comparing the power consumed by the different assignment algorithms for $\lambda = 30$ GCPS, $\alpha = 2$ and increasing values of x^* for Trace A (Synthetic traces).



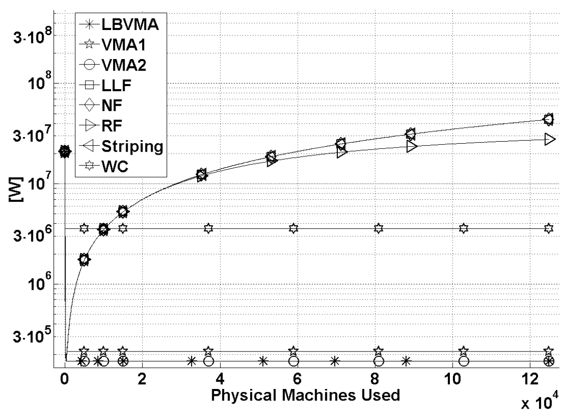
(a) $x^* = 10$ GCPS



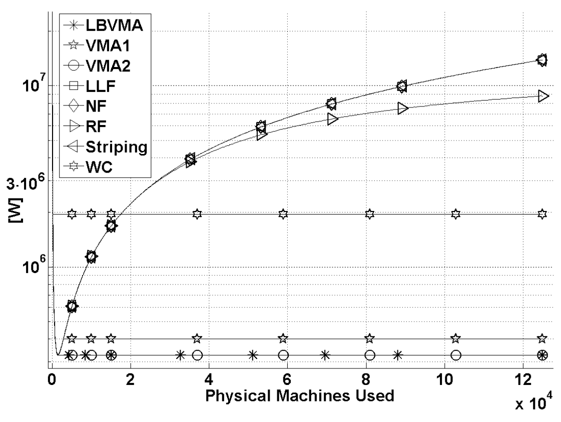
(b) $x^* = 100$ GCPS



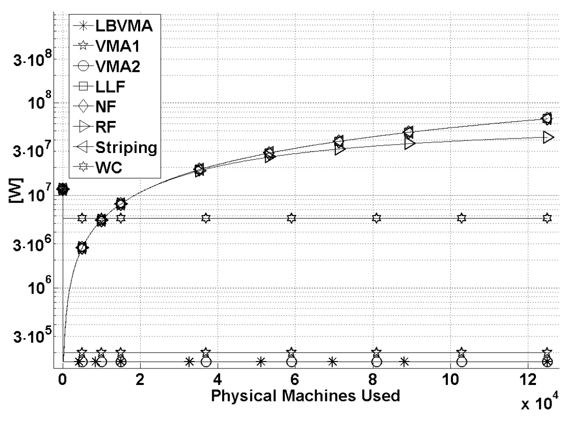
(c) $x^* = 30$ GCPS



(d) $x^* = 300$ GCPS



(e) $x^* = 50$ GCPS



(f) $x^* = 500$ GCPS

Figure 7.7: $(\cdot, m)$-VMA: Comparing the power consumed by the different assignment algorithms for $\lambda = 30$ GCPS, $\alpha = 2$ and increasing values of x^* for Trace B (Google traces).

Table 7.4: Simulation parameters for a set of machines with b and C similar to `Erdos`.

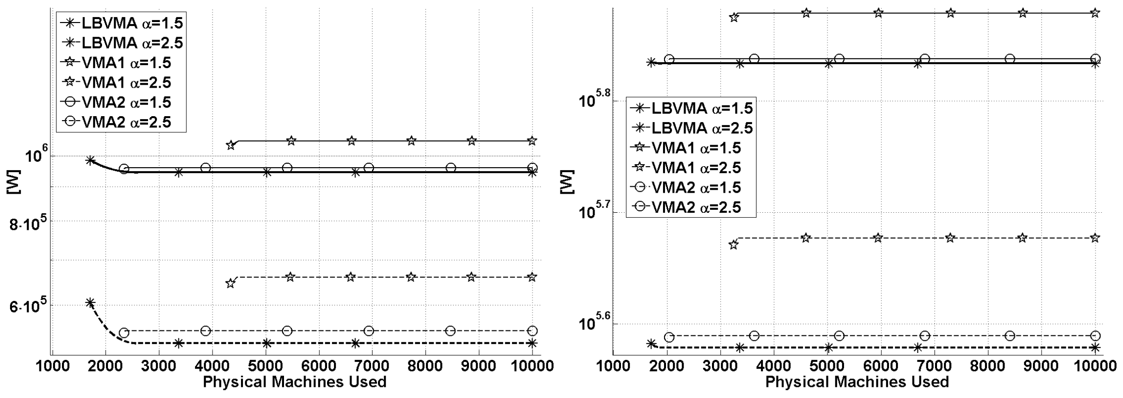
`Erdos`-like Family		Max. Capacity $C = 153.6$ GCPS			
		$b = 200$ W			
x^* [GCPS]	$x^*[\%C]$	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 2.5$	$\alpha = 3$
76.8	50	1.88E-14	3.39E-20	8.16E-26	2.21E-31
99.84	65	1.27E-14	2.01E-20	4.23E-26	1.00E-31
115.2	75	1.02E-14	1.51E-20	2.96E-26	6.54E-32
138.24	90	7.78E-15	1.05E-20	1.88E-26	3.79E-32
153.6	100	6.64E-15	8.48E-21	1.44E-26	2.76E-32
168.96	110	5.76E-15	7.01E-21	1.14E-26	2.07E-32

Table 7.5: Simulation parameters for a set of machines with b and C similar to `Nemesis`.

`Nemesis`-like Family		Max. Capacity $C = 11.2$ GCPS			
		$b = 80$ W			
x^* [GCPS]	$x^*[\%C]$	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 2.5$	$\alpha = 3$
5.6	50	3.82E-13	2.55E-18	2.27E-23	2.28E-28
7.28	65	2.58E-13	1.51E-18	1.18E-23	1.04E-28
8.4	75	2.08E-13	1.13E-18	8.25E-24	6.75E-29
10.08	90	1.58E-13	7.87E-19	5.23E-24	3.91E-29
11.2	100	1.35E-13	6.38E-19	4.02E-24	2.85E-29
12.32	110	1.17E-13	5.27E-19	3.17E-24	2.14E-29

Table 7.6: Simulation parameters for a set of machines with b and C similar to `Euler`.

`Euler`-like Family		Max. Capacity $C = 41.6$ GCPS			
$b = 80$ W					
x^* [GCPS]	$x^*[\%C]$	$\alpha = 1.5$	$\alpha = 2$	$\alpha = 2.5$	$\alpha = 3$
20.8	50	5.33E-14	1.85E-19	8.55E-25	4.44E-30
27.04	65	3.60E-14	1.09E-19	4.44E-25	2.02E-30
31.2	75	2.90E-14	8.22E-20	3.10E-25	1.32E-30
37.44	90	2.21E-14	5.71E-20	1.97E-25	7.62E-31
41.6	100	1.89E-14	4.62E-20	1.51E-25	5.56E-31
45.76	110	1.63E-14	3.82E-20	1.19E-25	4.17E-31
$b = 100$ W					
20.8	50	6.67E-14	2.31E-19	1.07E-24	5.56E-30
27.04	65	4.50E-14	1.37E-19	5.54E-25	2.53E-30
31.2	75	3.63E-14	1.03E-19	3.88E-25	1.65E-30
37.44	90	2.76E-14	7.13E-20	2.46E-25	9.53E-31
41.6	100	2.36E-14	5.78E-20	1.89E-25	6.95E-31
45.76	110	2.04E-14	4.78E-20	1.49E-25	5.22E-31
$b = 120$ W					
20.8	50	8.00E-14	2.77E-19	1.28E-24	6.67E-30
27.04	65	5.40E-14	1.64E-19	6.65E-25	3.03E-30
31.2	75	4.35E-14	1.23E-19	4.65E-25	1.98E-30
37.44	90	3.31E-14	8.56E-20	2.95E-25	1.14E-30
41.6	100	2.83E-14	6.93E-20	2.27E-25	8.33E-31
45.76	110	2.45E-14	5.73E-20	1.79E-25	6.26E-31



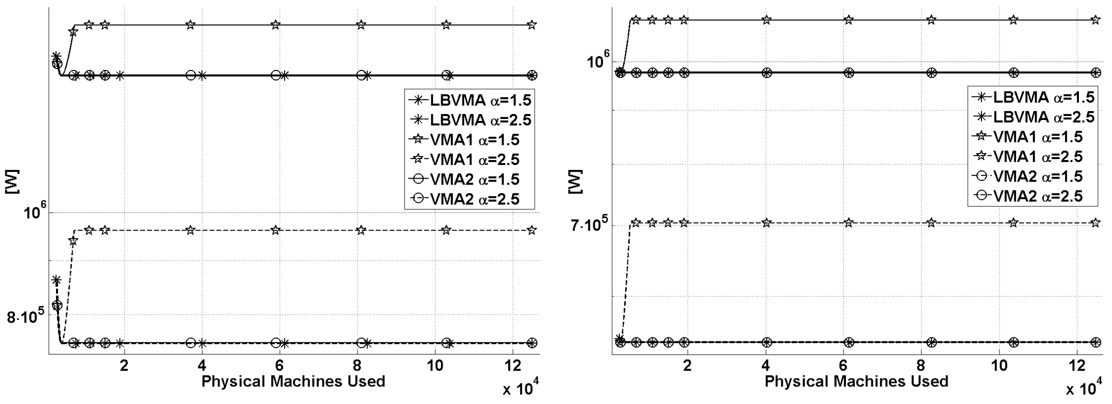
(a) $C = 153.6$ GCPS, $x^* = 0.65 \cdot C$ (b) $C = 153.6$ GCPS, $x^* = 0.9 \cdot C$

Figure 7.8: (C, m)-VMA: Comparing the power consumed by *VMA1* and *VMA2* with the lower bound *LBVMA* for $x^* = \{0.65, 0.9\} \cdot C$, $C = 153.6$ GCPS, $\alpha = \{1.5, 2.5\}$ for Trace A (synthetic traces).

compare again *VMA1* and *VMA2* with *LBVMA* for different values of α to evaluate its influence. In the second scenario, we evaluate the performance of *VMA1*, *VMA2*, and *LBVMA* jointly with *FF*, *MLF*, *Packing*, *RR*, *RF*, *LFF*, *NF*, *Striping* and *WC* when $alpha$, b and C are fixed and x^* varies. Finally, in the third scenario, we evaluate the influence of b on the performance of the different algorithms when α and C are fixed and different values of x^* and b are considered. Note that all experiments are run for both Trace A and Trace B. Similarly, observe that results are presented, again, as graphs in which the power consumed is represented as a function of the number of PMs used but, this time, these results do not start from 1 PM. Basically, each one of the results of the different algorithms starts from the number of PMs for which it obtained a valid solution and PMs where PMs do not have to be loaded beyond their capacity C.

The results for the first scenario, shown in Figures 7.8 and 7.9, throw very similar results to the ones obtained for $(\cdot, m)$-VMA. As for $(\cdot, m)$-VMA, *VMA2* performance is very close to *LBVMA* when it does not match it. Similarly, *VMA1* is again worse than *VMA2* due to the fact that it tends to pack less VMs per PM than *VMA1*.

The results for the second scenario are presented in Figure 7.10 and Figure 7.11 for Traces A and B, respectively. In them we compare the performance of *VMA1*, *VMA2*, *LBVMA*, *FF*, *MLF*, *Packing*, *RR*, *RF*, *LFF*, *NF*, *Striping* and *WC* for PMs such as the ones defined in Table 7.5. We can easily observe, independently of the trace used, that *RP*, *LFF*, *NF* and *Striping* consistently obtain partitions which result in a higher cost in Watts than the other algorithms. The nature of this set of algorithms implies using a large number of PMs with a small amount of load per PM, resulting, always, in a higher power consumption. For this reason, and for the sake of clarity when plotting and comparing the other algorithms, we only show *RP*, *LFF*, *NF* and *Striping* in subfigures 7.10(a), 7.10(b), 7.11(a) and 7.11(b) as an example. The rest of the subfigures from Figure 7.10 and Figure 7.11 zoom in the results of the other algorithms.



(a) $C = 153.6$ GCPS, $x^* = 0.65 \cdot C$ (b) $C = 153.6$ GCPS, $x^* = 0.9 \cdot C$

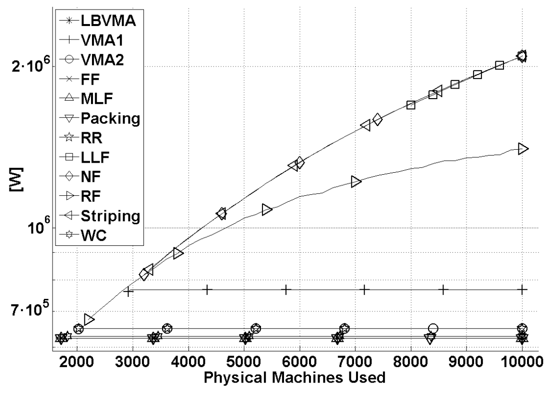
Figure 7.9: (C, m)-VMA: Comparing the power consumed by *VMA1* and *VMA2* with the lower bound *LBVMA* for $x^* = \{0.65, 0.9\} \cdot C$, $C = 153.6$ GCPS, $\alpha = \{1.5, 2.5\}$ for Trace B (Google traces).

Oppositely to the $(\cdot, m)$-VMA case, *WC* exhibits a better performance than *VMA1*, that also loses in the comparison with the group formed by *FF*, *MLF*, *Packing* and *RR* in every case except for Trace A and $x^* = 0.5 \cdot C$, shown in Figure 7.10(f)[4]. This worse performance of *VMA1* is a consequence of the ability of the other algorithms to obtain more packed solutions whose load is, additionally, closer to x^* than *VMA1*'s solution. On the other hand, *VMA2* results are similar or slightly worse than *FF*, *MLF*, *Packing* and *RR* for Trace A when when $x^* > 0.75C$ (Figures 7.10(a), 7.10(b) and 7.10(c)). However, *VMA2* still outperforms the other algorithms when $x^* \leq 0.75C$ (Figures 7.10(d),7.10(e) and 7.10(f)) or when the average VM task load is smaller, i.e., all the cases of Trace B. Once again, *VMA2* stays close to the optimal and, in general, performs better than the other algorithms.

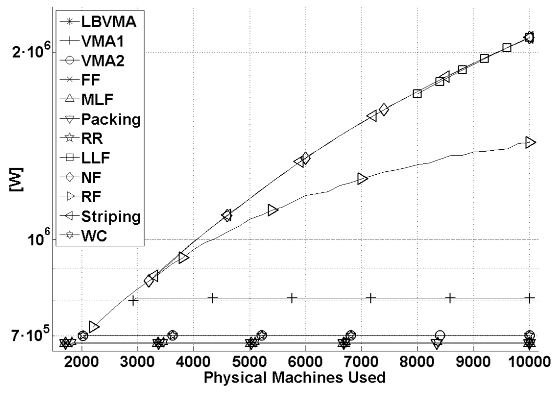
We finally evaluate the influence of b when fixing other parameters. These results are shown in Figures 7.12 and 7.13 for Traces A and B, respectively. We compare, again, the performance of *VMA1*, *VMA2*, *LBVMA*, *FF*, *MLF*, *Packing*, *RR*, *RF*, *LFF*, *NF*, *Striping* and *WC* when $\alpha = 2$, $C = 41.6$ GCPS, $x^* = \{0.65, 0.9\} \cdot C$ and $b = \{80, 100, 120\}$ W. Similarly to what happened in Scenario 2 (Figures 7.10 and 7.11), the results obtained by *RP*, *LFF*, *NF* and *Striping* are consistently worse than the ones obtained by the other algorithms. Therefore, as in the previous case, we only show them in the two first figures for each trace, namely Figures 7.12(a), 7.12(b), 7.13(a) and 7.13(b). The rest of the subfigures from Figures 7.12 and Figure 7.13 zoom in the results of the other algorithms.

Observing these results we can barely appreciate any difference in behavior/shape between the different algorithms when we x^* remains constant. Of course there are differences in the power consumed for each value of b, but this is mainly because of the contribution of b. The

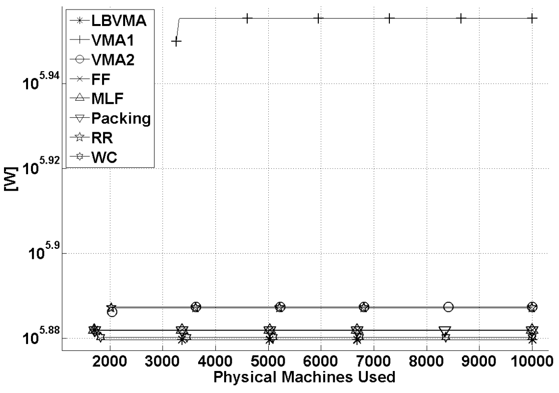
[4]This behaviour is not replicated for Trace B because of the smaller average load of Trace B VMs.



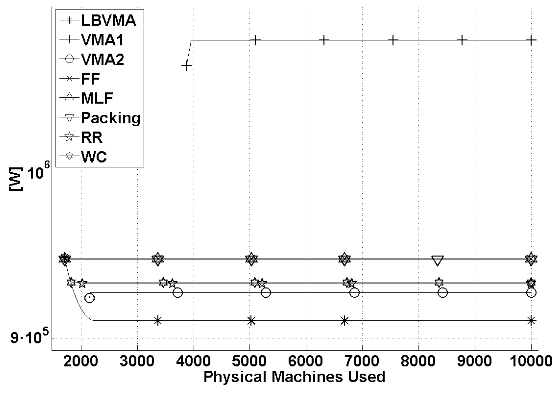
(a) $x^* = 1.1 \cdot C$



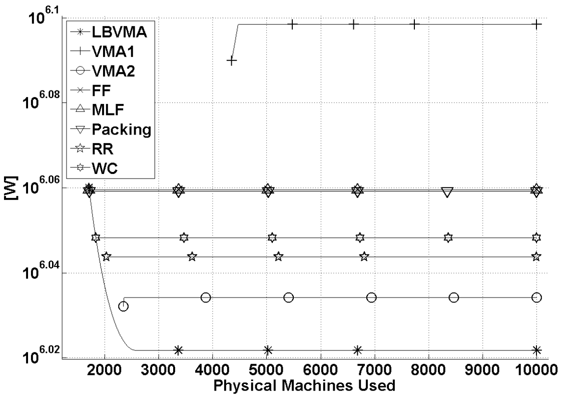
(b) $x^* = C$



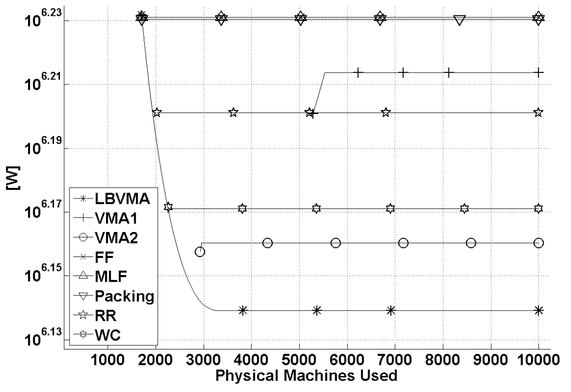
(c) $x^* = 0.9 \cdot C$



(d) $x^* = 0.75 \cdot C$

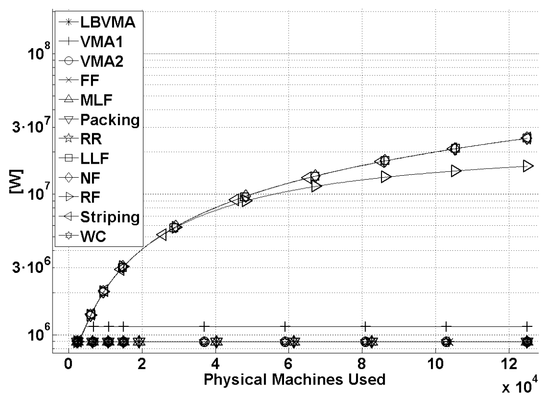


(e) $x^* = 0.65 \cdot C$

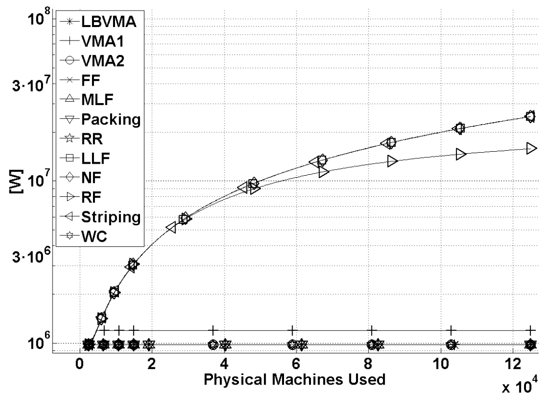


(f) $x^* = 0.5 \cdot C$

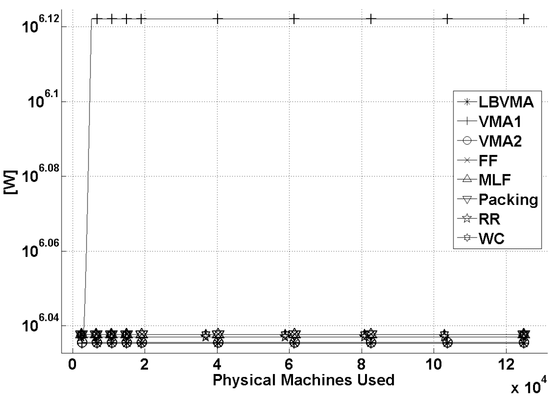
Figure 7.10: (C, m)-VMA: Comparing the power consumed by the different assignment algorithms for $C = 11.2$ GCPS, $\alpha = 2$ and different values of x^* for Trace A (Synthetic traces).



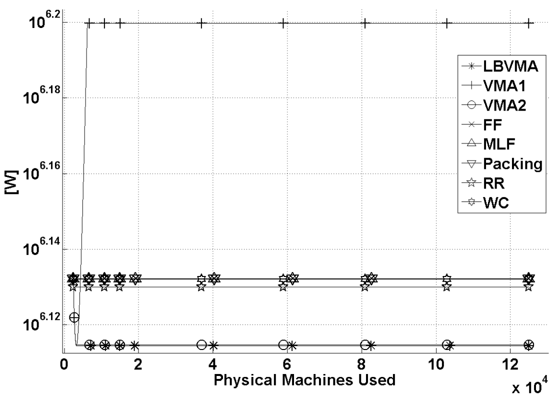
(a) $x^* = 1.1 \cdot C$



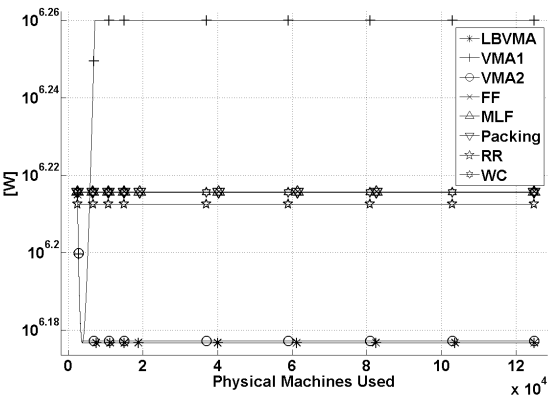
(b) $x^* = C$



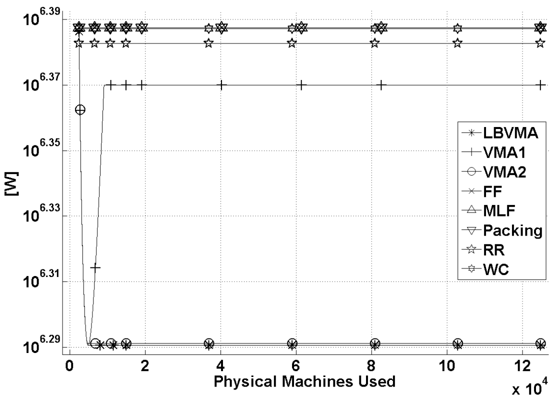
(c) $x^* = 0.9 \cdot C$



(d) $x^* = 0.75 \cdot C$

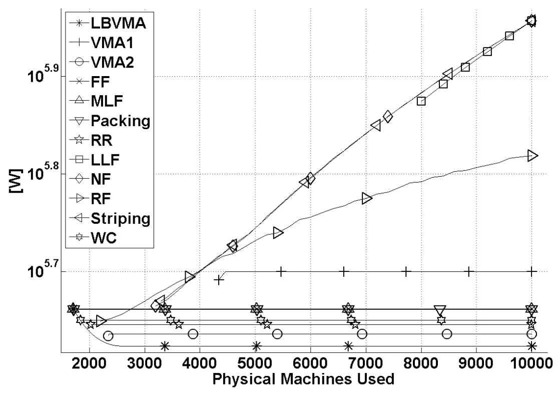


(e) $x^* = 0.65 \cdot C$

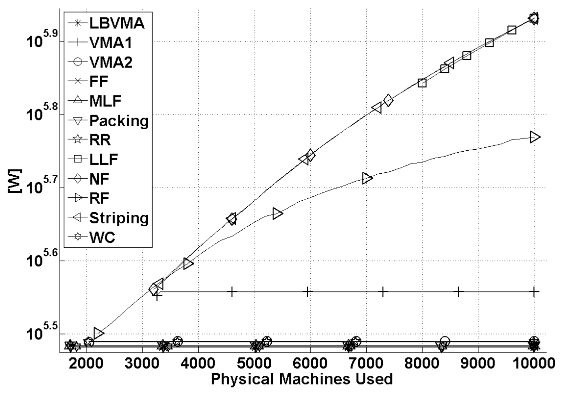


(f) $x^* = 0.5 \cdot C$

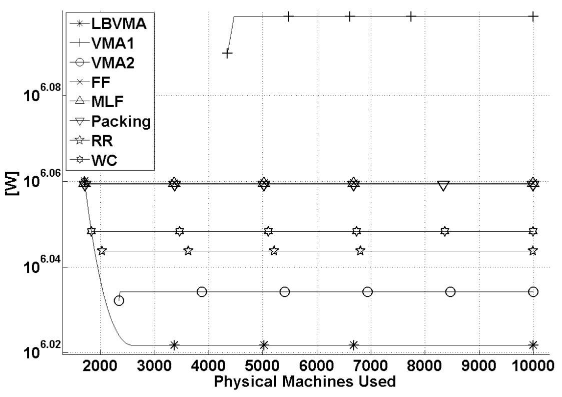
Figure 7.11: (C, m)-VMA: Comparing the power consumed by the different assignment algorithms for $C = 11.2$ GCPS, $\alpha = 2$ and different values of x^* for Trace B (Google traces).



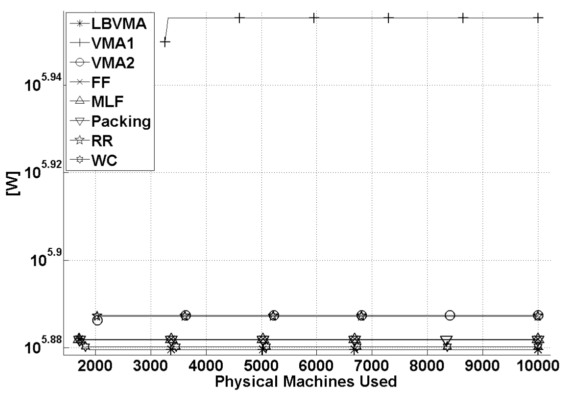
(a) $b = 80$ W, $x^* = 0.65 \cdot C$



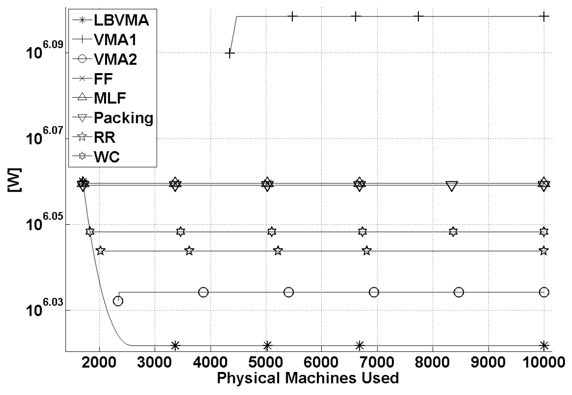
(b) $b = 80$ W, $x^* = 0.9 \cdot C$



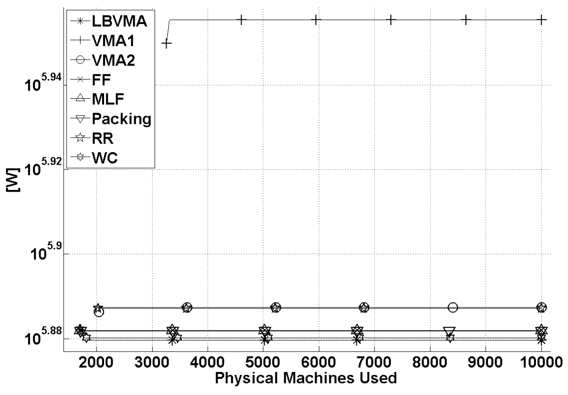
(c) $b = 100$ W, $x^* = 0.65 \cdot C$



(d) $b = 100$ W, $x^* = 0.9 \cdot C$

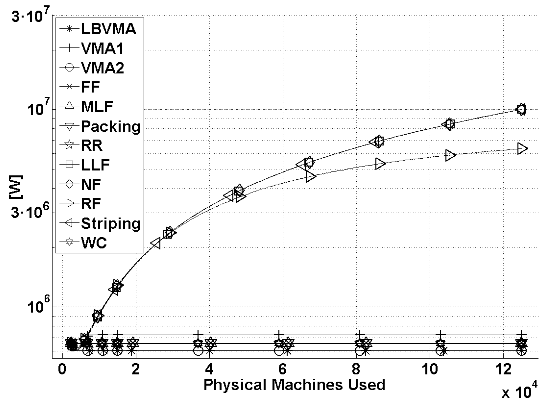


(e) $b = 120$ W, $x^* = 0.65 \cdot C$

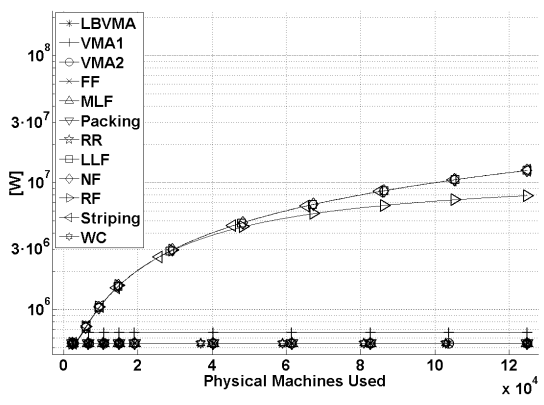


(f) $b = 120$ W, $x^* = 0.9 \cdot C$

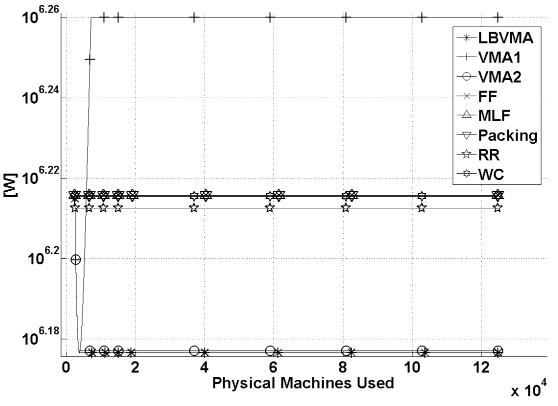
Figure 7.12: (C, m)-VMA: Comparing the power consumed by the different assignment algorithms fixing for $C = 41.6$ GCPS, $\alpha = 2$ and 2 different values of x^* while varying b for Trace A (Synthetic traces).



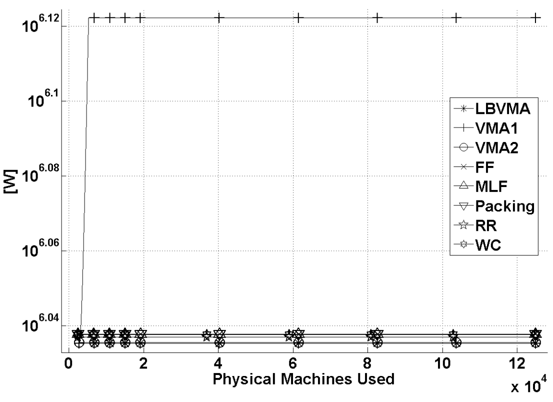
(a) $b = 80$ W, $x^* = 0.65 \cdot C$



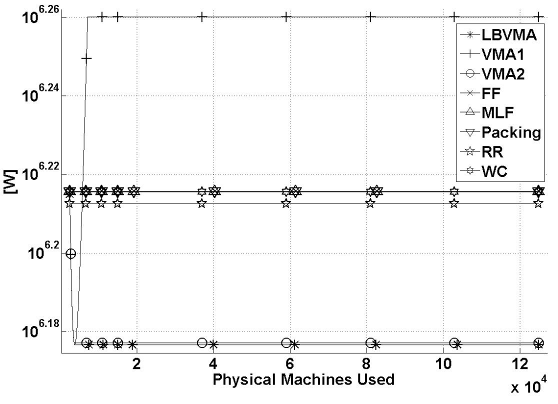
(b) $b = 80$ W, $x^* = 0.9 \cdot C$



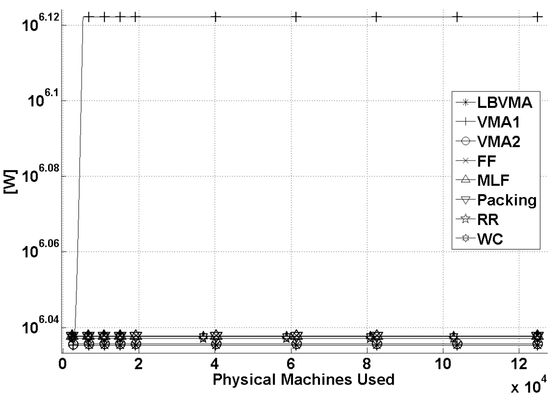
(c) $b = 100$ W, $x^* = 0.65 \cdot C$



(d) $b = 100$ W, $x^* = 0.9 \cdot C$



(e) $b = 120$ W, $x^* = 0.65 \cdot C$



(f) $b = 120$ W, $x^* = 0.9 \cdot C$

Figure 7.13: (C, m)-VMA: Comparing the power consumed by the different assignment algorithms for $C = 41.6$ GCPS, $\alpha = 2$ and 2 different values of x^* for Trace B (Google traces).

reason for these results is basically a limitation of our model when trying to emulate different PMs changing only the value of b. In this case, as μ depends also on b absorbs the changes on its value, resulting in an identical behavior when x^* is kept constant. The results are, therefore, similar to the ones achieved in the second scenario.

7.6 Discussion

We discuss in this section practical issues that must be addressed to apply our results to production environments.

Heterogeneity of Servers. For the sake of simplicity, we assume in our model that all servers in a data center are identical. We believe this is reasonable, considering that modern data centers are usually built with homogeneous commodity hardware. Nevertheless, the proposed model and derived results are also amenable to heterogeneous data center environments. In a heterogeneous data center, servers can be categorized into several groups with identical servers in each group. Then, different types of applications can be assigned to server groups according to their resource requirements. The VMA model presented here can be applied to the assignment problem of allocating tasks from the designated types of applications (especially CPU-intensive ones) to each group of servers. The approximation results we derive in this paper can be then combined with server-group assignment approximation bounds (out of the scope of our work) for energy-efficient task assignment in real data centers, regardless of the homogeneity of servers.

Consolidation. Traditionally, consolidation has been understood as a bin packing problem [123, 164], where VMs are assigned to PMs attempting to minimize the number of active PMs. However, the results we derived in this chapter, as well as the results shown in Chapter 6, show that such approach is not energy-efficient. Indeed, we showed that PM's should be loaded up to x^* to reduce energy consumption, even if this requires having more active PMs.

VM arrival and departure. When a new VM arrives to the system, or an assigned VM departs, adjustments to the assignment may improve energy efficiency. Given that the cost of VM migration is nowadays decreasing dramatically, our offline positive results can also be accommodated by reassigning VMs whenever the set of VM demands changes. Should the cost of migration be high to reassign after each VM arrival or departure, time could be divided in epochs buffering newly arrived VM demands until the beginning of the next epoch, when all (new and old) VMs would be reassigned (if necessary) running our offline approximation algorithm.

Multi-resource scheduling. This work focuses on CPU-intensive jobs (VMs) such as MapReduce-like tasks [59] which are representative in production datacenters. As the CPU is generally the dominant energy consumer in a server, assigning VMs according to CPU workloads entails energy efficiency. However, there exist types of jobs demanding heavily other computational resources, such as memory and/or storage. Although these resources have limited impact on a server's energy consumption, VMs performance may be degraded if they become the bottleneck resource in the system. In this case, a joint optimization of multiple resources (out of the scope of our work) is necessary for VMA.

Implementation on real systems. Most of the allocation algorithms that we have tested in Section 7.5 are already available in popular cloud platforms like OpenNebula [135] or Eucalyptus [68]. Including another allocation policy, such as our algorithms, in the cloud controllers of these and other platforms (e.g. Apache Mesos [18]) is feasible. Introducing our algorithms would make those platforms power efficient, providing power-aware allocation policies. This feature is not found on any of the algorithms tested in Section 7.5 except for the *Watts per Core* algorithm from [96]. We leave such integration for future work.

7.7 Conclusions

In this chapter, we have studied a particular case of the generalized assignment problem with applications to Cloud Computing. We have considered the problem of assigning virtual machines (VMs) to physical machines (PMs) so that the power consumption is minimized, a problem that we call virtual machine assignment (VMA). In our theoretical analysis, we have shown that the decision version of (C, m)-VMA problem is strongly NP-complete. We have shown as well that the $(C, \cdot)$-VMA, $(\cdot, m)$-VMA and $(\cdot, \cdot)$-VMA problems are strongly NP-hard. Hence, there is no FPTAS for these optimization problems. We have shown the existence of a PTAS that solves the $(\cdot, \cdot)$-VMA and $(\cdot, m)$-VMA offline problems. On the other hand, we have proved lower bounds on the approximation ratio of the $(C, \cdot)$-VMA and (C, m)-VMA problems. With respect to the online version of these problems, we have proved upper and lower bounds on the competitive ratio of the $(\cdot, \cdot)$-VMA, $(C, \cdot)$-VMA, $(\cdot, m)$-VMA, and (C, m)-VMA problems.

Chapter 8

Conclusions

Cloud computing is alive and probably no more than a toddler yet. However, even in this early age we are already able to see many of the advantages, surely not all, that cloud computing can offer us. Despite of all these advantages we must not forget about its drawbacks and this was one of the targets of this thesis.

The main objective of this thesis was to face two of the main problems of data centers, structural costs and energy consumption, which were introduced in Part I. Reducing the costs associated to these two issues is not only an economical target but also, most clearly in the case of energy consumption, an environmental problem. Power is generated at a cost and, indefinitely increasing our power consumption will have a prejudicial effect on our world. It is our duty, as researchers, to care about sustainability and ensure that future times will be better.

For these reasons we have provided of several tools to directly or indirectly tackle these problems. Part II was devoted to the case of structural costs. There, we provided, first, a way of computing the bisection (band)width of all the topologies that are or can be seen as product graphs. Bisection (band)width can help us to choose the most appropriate topology for our purposes when designing a data center. Choosing the right topology should result in a most efficient design of our network and probably lead to savings in deployment costs or energy savings.

Chapter 4 was also centered around reducing structural costs. In this case we proposed a heuristic that is able to achieve promising results for the Multidimensional Arrangement Problem (MAP). We can easily think of a data center as a 3-dimensional array or pseudo array where each one of the nodes is a server and the connections between these servers are the links of a graph. Therefore, mapping the graph that represents the data center topology to this 3-dimensional array representing the data center can be seen as a MAP. Our algorithm is useful to reduce the total length of the wires deployed, the length of the longest wire of a deployment, and also to simplify the deployment itself. Apart from the reduction on the deployment costs, it also implies a reduction on the energy wasted on the wires as well as a reduction on the latency.

Part III moved to the problem of energy consumption. Regarding this problem we proposed a different technique whose adoption would derive in a reduction of the energy consumption on data center networks, rate adaptation; studied how to reduce power consumption by optimizing how virtual machines are placed in physical machines, the virtual machine assignment problem; and finally studied how servers consume power and energy in a data center, characterizing the contribution of each component and confirming the superlinearity on the load that we assumed for both rate adaptation and virtual machine assignment.

Rate Adaptation, in Chapter 5, was showed to be able to achieve savings on the energy consumed by the network of more than a 20% on average, regardless of the topology employed. However, for fat tree topologies, probably the most common topology in nowadays data centers, it is able to achieve more than a 60% of instantaneous savings with the traces employed. We must bear in mind that we only considered rate adaptation, and did not combined it with powering down devices what, seeing the large amount of idle links when using our proposed algorithm, could result in a larger reduction of the energy consumed.

Moving then to servers, in Chapter 6 we characterized how different components of a data center server consumes power. One of the most interesting aspects, and also differentiator from previous works, is how we studied the effect of having multiple cores and how varying their frequency affects, not only to their energy consumption, but also to other components. The main idea to be extracted here is that a server is a puzzle, where each piece is a component, and some pieces needed others to perform a task, thus, being affected by how those other pieces are being operated. This study throw very interesting results as clearly confirming the superlinearity of power consumption on the load in the server as well as showing that the Active Cycles per Second are a proper unit to measure the load in a server. We concluded this part of the study showing how the characterization of these components can be used to predict the energy consumption of an application from its profiling.

Finally, taking advantage of some of the conclusions of the previous chapter, as the superlinearity of the power consumption on the load, we studied the virtual machine assignment problem and how such a superlinear power consumption model affects it. We thoroughly analyzed four different cases, depending on whether the capacity and the number of the servers where bounded or not. For each one of them, when possible, we provided upper and lower bounds on the approximation ratio, as well as upper and lower bounds on the competitive ratio of the algorithms we proposed for them. We also proved, by simulation, that the algorithms we proposed can obtain substantially cheaper solutions, from the point of view of power consumption of the system, than other algorithms proposed in the literature.

In general we proved that huge amounts of energy can still be saved in data centers with no need of updating the already deployed hardware, in some cases, and that new solutions are waiting for the new generation of servers and network devices, ready to optimize the way energy is used in data centers.

8.1 Future Work and Open Problems

As we said at the beginning of this section, Cloud Computing is still a toddler. This means that the amount of open problems is practically unlimited and that, usually, an answer bring up two more questions. Allow us, then, to focus only in the particular issues that we have worked at during this thesis.

Starting with Chapter 3, there are some particular and generic problems that remain open. The first one, regarding the bisection width of product graphs, computing the bisection width of graphs resulting from combining factor graphs with different normalized congestion or central cut remains open. One good example of these kind of graphs would be the cylinder, which results from combining paths and tori. Similarly, obtaining an exact result for the bisection bandwidth of the d-dimensional BCube remains as an open problem.

At a more general level, we believe that it would be interesting to explore the properties of product networks as topologies for data centers (currently, to our knowledge, only BCube fits in this category). The Cartesian product operation has been used in the past for building parallel processing topologies. We believe that it can be also used to build efficient topologies for data centers. Finally, another interesting problem would be to define more metrics that can help us to compare the newest data center topologies.

Regarding the Multidimensional Arrangement Problem presented in Chapter 4, the most obvious open problem is finding exact algorithms for large instances of graphs. While these exact algorithms are found, finding the amount of energy saved or the monetary reduction in deployment costs achieved by efficient deployments obtained by heuristics such as JAM, is a really interesting aspect to be studied. However, making such a comparison is complicated given the lack of information about deployment costs of real data centers or even regarding which are the topologies being used.

Although promising, rate adaptation cannot be deployed nowadays because of hardware limitations. We expect this open problem to be solved in the near future. In the meantime, studying how its combination with another techniques, such as powering down network devices, might reduce the energy consumption without degrading the performance of the network.

We are aware that the characterization of power consumption we presented in Chapter 6 is neither complete nor perfect. However, this is a key problem because the more accurately can we predict the power required by a server in a particular situation or the energy it will consume when a certain application is run, the better we will be able to assign tasks to servers or maneuver in case moving virtual machines across our servers is needed in order to increase the efficiency of our data center. In any case, we believe that some of the aspects we have proposed here will be helpful for future works. The most obvious open problems in this case are, first, to properly characterize the power consumption due to the RAM and, second, improve power characterizations so the accuracy is increased.

Finally, regarding the virtual machine assignment problem presented in Chapter 7, our future work will consider the possibility that the load incurred by a VM changes over time or that the assignment of VMs to PMs is not final (and VMs can migrate, maybe at a cost). In fact, if the migration of VMs is available for free, our offline positive results can also be used in these new models, since an offline approximation algorithm can be run each time a load changes or a new VM arrives. Then, the VMs can be redistributed accordingly. Another future extension of the model will consider that the power consumption of a feasible solution of the VMA problem depends on several parameters simultaneously (e.g., memory space or communication bandwidth, in addition to processing load). Finally, as stated in Section 6.5, we plan to deploy our algorithm in a cloud platform, probably OpenNebula, and compare the performance of our proposed algorithms against other state-of-the-art allocation algorithms such as the ones analyzed in Section 7.5.

References

[1] Amazon web services. http://aws.amazon.com. Accessed August 27, 2012.

[2] Cisco data center network topology. http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/DC_3_0/DC-3_0_IPInfra.html.

[3] Citrix. http://www.citrix.com. Accessed August 27, 2012.

[4] Infiniband. http://www.infinibandta.org.

[5] Rackspace. http://www.rackspace.com. Accessed August 27, 2012.

[6] U.s. environmental protection agency's data center report to congress. http://tinyurl.com/2jz3ft.

[7] Emile H.L. Aarts and P.J.M. van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips Journal of Research*, 40(4):193, 1985.

[8] Dennis Abts, Michael R. Marty, Philip M. Wells, Peter Klausler, and Hong Liu. Energy proportional datacenter networks. In *ISCA*, pages 338–347, 2010.

[9] Gerald Aigner and William H Whitted. Modular data center, October 9 2007. US Patent 7,278,273.

[10] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 63–74. ACM, 2008.

[11] Noga Alon, Yossi Azar, Gerhard J. Woeginger, and Tal Yadid. Approximation schemes for scheduling. In Michael E. Saks, editor, *SODA*, pages 493–500. ACM/SIAM, 1997.

[12] Noga Alon, Yossi Azar, Gerhard J Woeginger, and Tal Yadid. Approximation schemes for scheduling on parallel machines. *Journal of Scheduling*, 1(1):55–66, 1998.

[13] Matthew Andrews, Antonio Fernández Anta, Lisa Zhang, and Wenbo Zhao. Routing for power minimization in the speed scaling model. *IEEE/ACM Trans. Netw.*, 20(1):285–294, 2012.

[14] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Minimum-cost network design with (dis)economies of scale. In *Proc. of 51-st Annual IEEE Symposium on Foundations of Computer Science*, pages 585–592, 2010.

[15] Matthew Andrews, Spyridon Antonakopoulos, and Lisa Zhang. Energy-aware scheduling algorithms for network stability. In *INFOCOM*, pages 1359–1367, 2011.

[16] Matthew Andrews, Antonio Fernández, Lisa Zhang, and Wenbo Zhao. Routing and scheduling for energy and delay minimization in the powerdown model. In *INFOCOM*, pages 21–25, 2010.

[17] Antonio Antoniadis, Sungjin Im, Ravishankar Krishnaswamy, Benjamin Moseley, Viswanath Nagarajan, Krik Pruhs, and Cliff Stein. Hallucination helps: Energy efficient circuit routing. In Erlebach and Persiano [67].

[18] Apache. Apache mesos. `http://http://mesos.apache.org/`, 2014. Accessed December 11th, 2014.

[19] M. Cemil Azizoğlu and Ömer Eğecioğlu. The isoperimetric number and the bisection width of generalized cylinders. *Electronic Notes in Discrete Mathematics*, 11:53–62, 2002.

[20] M. Cemil Azizoğlu and Ömer Eğecioğlu. Extremal sets minimizing dimension-normalized boundary in hamming graphs. *SIAM J. Discrete Math.*, 17(2):219–236, 2003.

[21] M. Cemil Azizoğlu and Ömer Eğecioğlu. The bisection width and the isoperimetric number of arrays. *Discrete Applied Mathematics*, 138(1-2):3–12, 2004.

[22] M Bailey, M Eastwood, T Grieser, L Borovick, V Turner, and RC Gray. Special study: Data center of the future, 2007.

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

[24] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proc. of 20-th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 693–701, 2009.

[25] Nikhil Bansal, Anupam Gupta, Ravishankar Krishnaswamy, Viswanath Nagarajan, Kirk Pruhs, and Cliff Stein. Multicast routing for energy minimization using speed scaling. In *MedAlg*, pages 37–51, 2012.

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

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

[28] Luiz André Barroso and Urs Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines, 2nd edition. *Synthesis lectures on computer architecture*, 2013.

[29] Robert Basmadjian, Nasir Ali, Florian Niedermeier, Hermann de Meer, and Giovanni Giuliani. A methodology to predict the power consumption of servers in data centres. In *ACM e-Energy*, pages 1–10, 2011.

[30] Robert Basmadjian and Hermann de Meer. Evaluating and modeling power consumption of multi-core processors. In *Third International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, pages 1–10. IEEE, 2012.

[31] CL Belady. Projecting annual new datacenter construction market size. *Microsoft, Global Foundation Services Report*, 2011.

[32] Umesh Bellur, Chetan S. Rao, and Madhu Kumar SD. Optimal placement algorithms for virtual machines. *CoRR*, abs/1011.5064, 2010.

[33] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28(5):755–768, 2012.

[34] Una Benlic and Jin-Kao Hao. Breakout local search for the quadratic assignment problem. *Applied Mathematics and Computation*, 219(9):4800–4815, 2013.

[35] Theophilus Benson, Ashok Anand, Aditya Akella, and Ming Zhang. Understanding data center traffic characteristics. *Computer Communication Review*, 40(1):92–99, 2010.

[36] Bruno Biais and Paul Woolley. High frequency trading. *Manuscript, Toulouse University, IDEI*, 2011.

[37] Juan Felipe Botero, Xavier Hesselbach, Michael Duelli, Daniel Schlosser, Andreas Fischer, and Hermann de Meer. Energy efficient virtual network embedding. *IEEE Communications Letters*, 16(5):756–759, 2012.

[38] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[39] Alaa Brihi and Waltenegus Dargie. Dynamic voltage and frequency scaling in multimedia servers. In *IEEE AINA*, 2013.

[40] David Brooks, Pradip Bose, Stanley Schuster, Hans M. Jacobson, Prabhakar Kudva, Alper Buyuktosunoglu, John-David Wellman, Victor V. Zyuban, Manish Gupta, and Peter W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[41] Rainer E. Burkard, Stefan E. Karisch, and Franz Rendl. Qaplib–a quadratic assignment problem library. *Journal of Global Optimization*, 10(4):391–403, 1997.

[42] R.E. Burkard and F. Rendl. A thermodynamically motivated simulation procedure for combinatorial optimization problems. *European Journal of Operational Research*, 17(2):169 – 174, 1984.

[43] Alberto Caprara, Adam N. Letchford, and Juan-José Salazar-González. Decorous lower bounds for minimum linear arrangement. *INFORMS Journal on Computing*, 23(1):26–40, 2011.

[44] Alberto Caprara, Marcus Oswald, Gerhard Reinelt, Robert Schwarz, and Emiliano Traversi. Optimal linear arrangements using betweenness variables. *Mathematical Programming Computation*, 3(3):261–280, 2011.

[45] M. Cardosa, A. Singh, H. Pucha, and A. Chandra. Exploiting spatio-temporal tradeoffs for energy-aware mapreduce in the cloud. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 251 –258, 2011.

[46] P Castagna. Having fun with pagerank and mapreduce. *Hadoop User Group UK talk. Available: http://static. last. fm/johan/huguk-20090414/paolo_castagna-pagerank. pdf.*

[47] Deeparnab Chakrabarty, Chandra Chekuri, Sanjeev Khanna, and Nitish Korula. Approximability of capacitated network design. In *IPCO*, pages 78–91, 2011.

[48] Ashok K. Chandra and C. K. Wong. Worst-case analysis of a placement algorithm related to storage allocation. *SIAM J. Comput.*, 4(3):249–263, 1975.

[49] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 103–116, 2001.

[50] Shih-Chang Chen, Chih-Chun Lee, Hsi-Ya Chang, Kuan-Chou Lai, Kuan-Ching Li, and Chunming Rong. Energy-aware task consolidation technique for cloud computing. In *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, pages 115–121, 2011.

[51] Susanta Nanda Tzi-cker Chiueh and Stony Brook. A survey on virtualization technologies. *RPE Report*, pages 1–42, 2005.

[52] Michael Chlistalla, Bernhard Speyer, Sabine Kaiser, and Thomas Mayer. High-frequency trading. *Deutsche Bank Research*, pages 1–19, 2011.

[53] Antonio Cianfrani, Vincenzo Eramo, Marco Listanti, Marco Polverini, and Athanasios V. Vasilakos. An ospf-integrated routing strategy for qos-aware energy saving in ip backbone networks. *IEEE Transactions on Network and Service Management*, 9(3):254–267, 2012.

[54] R. A. Cody and Edward G. Coffman Jr. Record allocation for minimizing expected retrieval costs on drum-like storage devices. *J. ACM*, 23(1):103–115, 1976.

[55] David T. Connolly. An improved annealing scheme for the qap. *European Journal of Operational Research*, 46(1):93–100, 1990.

[56] Standard Performance Evaluation Corporation. Spec power benchmark, 2007.

[57] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[58] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Computers*, 39(6):775–785, 1990.

[59] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[60] Zvi Drezner. Compounded genetic algorithms for the quadratic assignment problem. *Oper. Res. Lett.*, 33(5):475–480, 2005.

[61] Jose Duato, Sudhakar Yalamanchili, and Ni Lionel. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[62] Dimitris Economou, Suzanne Rivoire, Christos Kozyrakis, and Partha Ranganathan. Full-system power analysis and modeling for server environments. In *Proceedings of Workshop on Modeling, Benchmarking, and Simulation*, pages 70–77, 2006.

[63] Kemal Efe and Gui-Liang Feng. A proof for bisection width of grids. *World Academy of Science, Engineering and Technology*, 27(31):172 – 177, 2007.

[64] Kemal Efe and Antonio Fernández. Products of networks with logarithmic diameter and fixed degree. *IEEE Trans. Parallel Distrib. Syst.*, 6(9):963–975, 1995.

[65] Kemal Efe and Antonio Fernández. Mesh-connected trees: A bridge between grids and meshes of trees. *IEEE Trans. Parallel Distrib. Syst.*, 7(12):1281–1291, 1996.

[66] Leah Epstein and Jiri Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. *Algorithmica*, 39(1):43–57, 2004.

[67] Thomas Erlebach and Giuseppe Persiano, editors. *Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*, volume 7846 of *Lecture Notes in Computer Science*. Springer, 2013.

[68] Eucalyptus. Eucalyptus. `http://www.eucalyptus.com/`. Accessed January 20th, 2013.

[69] W. Fernandez de la Vega and G.S. Lueker. Bin packing can be solved within $1 + \epsilon$ in linear time. *Combinatorica*, 1(4):349–355, 1981.

[70] Matteo Fischetti, Michele Monaci, and Domenico Salvagnin. Three ideas for the quadratic assignment problem. *Operations Research*, 60(4):954–964, 2012.

[71] Charles Fleurent and Jacques A. Ferland. Genetic hybrids for the quadratic assignment problem. In *DIMACS Series in Mathematics and Theoretical Computer Science*, pages 173–187. American Mathematical Society, 1993.

[72] Charles Fleurent and Fred Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11(2):198–204, 1999.

[73] Anshul Gandhi, Mor Harchol-Balter, Rajarshi Das, and Charles Lefurgy. Optimal power allocation in server farms. In *ACM SIGMETRICS*, pages 157–168, 2009.

[74] Andres Garcia-Saavedra, Pablo Serrano, Albert Banchs, and Giuseppe Bianchi. Energy consumption anatomy of 802.11 devices and its implication on modeling and design. In *ACM CoNEXT*, pages 169–180, 2012.

[75] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[76] Rong Ge, Xizhou Feng, Shuaiwen Song, Hung-Ching Chang, Dong Li, and Kirk W Cameron. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on*, 21(5):658–671, 2010.

[77] George Ginis. Low-power modes for adsl2 and adsl2+. *White Paper, Broadband Communications Group, Texas Instruments*, Jan. 2005.

[78] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: a scalable and flexible data center network. *SIGCOMM Comput. Commun. Rev.*, 39:51–62, August 2009.

[79] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. *SIGCOMM Comput. Commun. Rev.*, 39:63–74, August 2009.

[80] Chuanxiong Guo, Haitao Wu, Kun Tan, Lei Shi, Yongguang Zhang, and Songwu Lu. Dcell: a scalable and fault-tolerant network structure for data centers. *SIGCOMM Comput. Commun. Rev.*, 38:75–86, August 2008.

[81] Deke Guo, Tao Chen, Dan Li, Mo Li, Yunhao Liu, and Guihai Chen. Expandable and cost-effective network structures for data centers using dual-port servers. *Computers, IEEE Transactions on*, 62(7):1303–1317, 2013.

[82] Anupam Gupta, Ravishankar Krishnaswamy, and Kirk Pruhs. Online primal-dual for non-linear optimization with applications to speed scaling. In Erlebach and Persiano [67], pages 173–186.

[83] Maruti Gupta and Suresh Singh. Greening of the internet. In *SIGCOMM*, pages 19–26, 2003.

[84] Maruti Gupta and Suresh Singh. Using low-power modes for energy conservation in ethernet lans. In *INFOCOM*, pages 2451–2455, 2007.

[85] Mark D. Hansen. Approximation algorithms for geometric embeddings in the plane with applications to parallel processing problems. In *FOCS, 1989., 30th Annual Symposium on*, pages 604–609. IEEE, 1989.

[86] Ward Van Heddeghem, Sofie Lambert, Bart Lannoo, Didier Colle, Mario Pickavet, and Piet Demeester. Trends in worldwide ICT electricity consumption from 2007 to 2012. *Computer Communications*, Submitted.

[87] Brandon Heller, Srini Seetharaman, Priya Mahadevan, Yiannis Yiakoumis, Puneet Sharma, Sujata Banerjee, and Nick McKeown. Elastictree: saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 249–264, Berkeley, CA, USA, 2010. USENIX Association.

[88] Joseph L. Hellerstein. Google cluster data. Google research blog, January 2010. Posted at `http://googleresearch.blogspot.com/2010/01/google-cluster-data.html`.

[89] Lei Huang, Qin Jia, Xin Wang, Shuang Yang, and Baochun Li. Pcube: Improving power efficiency in data center networks. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 65–72. IEEE, 2011.

[90] Lei Huang, Qin Jia, Xin Wang, Shuang Yang, and Baochun Li. Pcube: Improving power efficiency in data center networks. In *IEEE CLOUD*, pages 65–72, 2011.

[91] IBM. Ibm's integrated server room. `http://www-935.ibm.com/services/in/en/it-services/integrated-server-room.html`.

[92] IEEE Std. 802.3az. Energy Efficient Ethernet, 2010.

[93] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. Online scheduling with general cost functions. In *Proc. of 23-rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1254–1265, 2012.

[94] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM TALG*, 3(4):41, 2007.

[95] Tabitha James, César Rego, and Fred Glover. Multistart tabu search and diversification strategies for the quadratic assignment problem. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Trans. on*, 39(3):579–596, 2009.

[96] R. Jansen and P.R. Brenner. Energy efficient virtual machine allocation in the cloud. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1–8, 2011.

[97] D N Jayasimha, B Zafar, and Y Hoskote. On chip interconnection networks why they are different and how to compare them. *Intel*, 2006.

[98] Zhu Jingwei, Rui Ting, Fang Husheng, Zhang Jinlin, and Liao Ming. Simulated annealing ant colony algorithm for qap. In *ICNC 2012*, pages 789–793, 2012.

[99] Scott Kirkpatrick, D. Gelatt Jr., and Mario P Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[100] Tjalling C. Koopmans and Martin Beckmann. Assignment problems and the location of economic activities. *Econometrica: Journal of the Econometric Society*, pages 53–76, 1957.

[101] Bhavani Krishnan, Hrishikesh Amur, Ada Gavrilovska, and Karsten Schwan. VM power metering: feasibility and challenges. *ACM SIGMETRICS Performance Evaluation Review*, 38(3):56–60, 2011.

[102] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster computing*, 12(1):1–15, 2009.

[103] Dara Kusic, Jeffrey O. Kephart, James E. Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[104] Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems*, pages 1–8. USENIX Association, 2010.

[105] F. Thomson Leighton. *Introduction to parallel algorithms and architectures: array, trees, hypercubes*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[106] Adam Wade Lewis, Soumik Ghosh, and Nian-Feng Tzeng. Run-time energy consumption estimation based on workload in server systems. *HotPower'08*, pages 17–21, 2008.

[107] Yong Li, Panos M. Pardalos, and Mauricio G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. *Quadratic Assignment and Related Problems*, 16:237–261, 1994.

[108] Chenguang Liu, Jianzhong Huang, Qiang Cao, Shenggang Wan, and Changsheng Xie. Evaluating energy and performance for server-class hardware configurations. In *IEEE NAS*, pages 339–347, 2011.

[109] Ning Liu, Ziqian Dong, and Roberto Rojas-Cessa. Task and server assignment for reduction of energy consumption in datacenters. In *Proceedings of the IEEE 11-th International Symposium on Network Computing and Applications*, pages 171–174, 2012.

[110] F. Machida, M. Kawato, and Y. Maeno. Redundant virtual machine placement for fault-tolerant consolidated server clusters. In *Network Operations and Management Symposium (NOMS), 2010 IEEE*, pages 32 –39, 2010.

[111] Priya Mahadevan, Sujata Banerjee, Puneet Sharma, Amip Shah, and Parthasarathy Ranganathan. On energy efficiency for enterprise and data center networks. *IEEE Communications Magazine*, 49(8):94–100, 2011.

[112] C.C.T. Mark, D. Niyato, and Tham Chen-Khong. Evolutionary optimal virtual machine placement and demand forecaster for cloud computing. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pages 348 –355, 2011.

[113] Andrew J. Mcallister. A new heuristic algorithm for the linear arrangement problem. Technical Report TR-99-126a, University of New Brunswick, 1999.

[114] Xiaoqiao Meng, Vasileios Pappas, and Li Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *INFOCOM*, pages 1154–1162, 2010.

[115] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087, 1953.

[116] K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, pages 91–98, 2011.

[117] Patrick Mills, Edward Tsang, and John Ford. Applying an extended guided local search to the quadratic assignment problem. *Annals of Operations Research*, 118(1-4):121–135, 2003.

[118] M. Mirza-Aghatabar, S. Koohi, S. Hessabi, and M. Pedram. An empirical investigation of mesh and torus noc topologies under different routing algorithms and traffic models. In *Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools*, pages 19–26, Washington, DC, USA, 2007. IEEE Computer Society.

[119] Alfonsas Misevičius. A modified simulated annealing algorithm for the quadratic assignment problem. *Informatica*, 14(4):497–514, 2003.

[120] Alfonsas Misevičius. An improved hybrid genetic algorithm: new results for the quadratic assignment problem. *Knowl.-Based Syst.*, 17(2-4):65–73, 2004.

[121] Alfonsas Misevičius. A tabu search algorithm for the quadratic assignment problem. *Comp. Opt. and Appl.*, 30(1):95–111, 2005.

[122] Alfonsas Misevičius. An implementation of the iterated tabu search algorithm for the quadratic assignment problem. *OR Spectrum*, 34(3):665–690, 2012.

[123] M. Mishra and A. Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 275 –282, 2011.

[124] Mayank Mishra and Anirudha Sahoo. On theory of vm placement: Anomalies in existing methodologies and their mitigation using a novel vector based approach. In *IEEE CLOUD*, pages 275–282, 2011.

[125] Akihiko Miyoshi, Charles Lefurgy, Eric Van Hensbergen, Ram Rajamony, and Raj Rajkumar. Critical power slope: understanding the runtime effects of frequency scaling. In *ACM ICS'02*, pages 35–44, 2002.

[126] Justin D Moore, Jeffrey S Chase, Parthasarathy Ranganathan, and Ratnesh K Sharma. Making scheduling "cool": Temperature-aware workload placement in data centers. In *USENIX annual technical conference, General Track*, pages 61–75, 2005.

[127] Koji Nakano. Linear layout of generalized hypercubes. *Int. J. Found. Comput. Sci.*, 14(1):137–156, 2003.

[128] Ripal Nathuji and Karsten Schwan. Virtualpower: coordinated power management in virtualized enterprise systems. In *SOSP*, pages 265–278, 2007.

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

[130] Hien Nguyen Van, Frederic Dang Tran, and Jean-Marc Menaud. Autonomic virtual resource management for service hosting platforms. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 1–8. IEEE Computer Society, 2009.

[131] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39:39–50, August 2009.

[132] Volker Nissen and Henrik Paul. A modification of threshold accepting and its application to the quadratic assignment problem. *Operations-Research-Spektrum*, 17(2-3):205–210, 1995.

[133] Bruce Nordman and K Christensen. Reducing the energy consumption of network devices. *IEEE 802.3 tutorial*, pages 1–30, 2005.

[134] Axel Nyberg, Tapio Westerlund, and Andreas Lundell. Improved discrete reformulations for the quadratic assignment problem. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 193–203. Springer, 2013.

[135] OpenNebula. Opennebula. `http://opennebula.org/`. Accessed January 20th, 2013.

[136] Yi Pan, S. Q. Zheng, Keqin Li, and Hong Shen. An improved generalization of mesh-connected computers with multiple buses. *IEEE Trans. Parallel Distrib. Syst.*, 12:293–305, March 2001.

[137] Jordi Petit. Experiments on the minimum linear arrangement problem. *Journal of Experimental Algorithmics (JEA)*, 8:2–3, 2003.

[138] Ramya Raghavendra, Parthasarathy Ranganathan, Vanish Talwar, Zhikui Wang, and Xiaoyun Zhu. No power struggles: Coordinated multi-level power management for the data center. In *ACM SIGARCH Computer Architecture News*, volume 36, pages 48–59. ACM, 2008.

[139] Eduardo Rodríguez-Tello, Jin-Kao Hao, and Jose Torres-Jiménez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008.

[140] Eduardo Rodríguez-Tello and Jose Torres-Jiménez. A refined evaluation function for the minla problem. In *MICAI 2006*, pages 392–403, 2006.

[141] José D. P. Rolim, Ondrej Sýkora, and Imrich Vrto. Optimal cutwidths and bisection widths of 2- and 3-dimensional meshes. In Manfred Nagl, editor, *WG*, volume 1017 of *Lecture Notes in Computer Science*, pages 252–264. Springer, 1995.

[142] Ilya Safro, Dorit Ron, and Achi Brandt. Graph minimum linear arrangement by multilevel weighted edge contractions. *Journal of Algorithms*, 60(1):24–41, 2006.

[143] Sartaj Sahni and Teófilo F. González. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

[144] Erno Salminen, Ari Kulmala, and Timo D H. Survey of network-on-chip proposals. *Simulation*, (March):1–13, 2008.

[145] Steve Scott, Dennis Abts, John Kim, and William J. Dally. The blackwidow high-radix clos network. In *ISCA*, pages 16–28, 2006.

[146] Yunfei Shang, Dan Li, and Mingwei Xu. Energy-aware routing in data center network. In *Green Networking*, pages 1–8, 2010.

[147] Shekhar Srikantaiah, Aman Kansal, and Feng Zhao. Energy aware consolidation for cloud computing. In *Proceedings of the 2008 conference on Power aware computing and systems*, volume 10 of *HotPower'08*. USENIX Association, 2008.

[148] Matt Stansberry. Important to recognize the dramatic improvement in data center efficiency, 2012.

[149] Thomas Stützle. Max-min ant system for quadratic assignment problems. Technical Report Forschungsbericht AIDA-97-04, TU Darmstadt, 1997.

[150] Éric D. Taillard. Robust taboo search for the quadratic assignment problem. *Parallel Computing*, 17(4-5):443–455, 1991.

[151] Éric D. Taillard and Luca Maria Gambardella. Adaptive memories for the quadratic assignment problems. Technical report, 1997.

[152] Wendy Torell. Tco analysis of a traditional data center vs. a scalable, containerized data center. Technical Report 164, Schneider Electric, 2012.

[153] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove. Cost-efficient scheduling heuristics for deadline constrained workloads on hybrid clouds. In *Proceedings of the IEEE Third International Conference on Cloud Computing Technology and Science*, pages 320–327, 2011.

[154] James M. Varanelli and James P. Cohoon. A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Computers & Operations Research*, 26(5):481–503, 1999.

[155] Arunchandar Vasan, Anand Sivasubramaniam, Vikrant Shimpi, T Sivabalan, and Rajesh Subbiah. Worth their Watts? - An empirical study of datacenter servers. In *IEEE HPCA*, pages 1–10, 2010.

[156] Nedeljko Vasic, Prateek Bhurat, Dejan M. Novakovic, Marco Canini, Satyam Shekhar, and Dejan Kostic. Identifying and using energy-critical paths. In *CoNEXT*, page 18, 2011.

[157] Nedeljko Vasic and Dejan Kostic. Energy-aware traffic engineering. In *e-Energy*, pages 169–178, 2010.

[158] Vijay V. Vazirani. *Approximation Algorithms.* Springer, March 2004.

[159] H. Viswanathan, E.K. Lee, I. Rodero, D. Pompili, M. Parashar, and M. Gamell. Energy-aware application-centric vm allocation for hpc workloads. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 890 –897, 2011.

[160] M Mitchell Waldrop. Data center in a box. *Scientific American*, 297(2):90–93, 2007.

[161] Jiunn-Chin Wang. Solving quadratic assignment problems by a tabu based simulated annealing algorithm. In *ICIAS 2007*, pages 75–80. IEEE, 2007.

[162] Jiunn-Chin Wang. A multistart simulated annealing algorithm for the quadratic assignment problem. In *IBICA 2012*, pages 19–23. IEEE, 2012.

[163] Lin Wang, Antonio Fernández Anta, Fa Zhang, Chenying Hou, and Zhiyong Liu. Routing for energy minimization with discrete cost functions. *CoRR*, abs/1302.0234, 2013.

[164] Meng Wang, Xiaoqiao Meng, and Li Zhang. Consolidating virtual machines with dynamic bandwidth demand in data centers. In *IEEE INFOCOM*, pages 71–75, 2011.

[165] Xiaodong Wang, Yanjun Yao, Xiaorui Wang, Kefa Lu, and Qing Cao. Carpo: Correlation-aware power optimization in data center networks. In *INFOCOM*, pages 1125–1133, 2012.

[166] Mark Weiser, Brent Welch, Alan Demers, and Scott Shenker. Scheduling for reduced CPU energy. In *Mobile Computing*, pages 449–471. Springer, 1996.

[167] Mickey R. Wilhelm and Thomas L. Ward. Solving quadratic assignment problems by simulated annealing. *IIE Transactions*, 19(1):107–119, 1987.

[168] Jing Xu and José Fortes. A multi-objective approach to virtual machine management in datacenters. In *Proceedings of the 8th ACM international conference on Autonomic computing*, ICAC '11, pages 225–234. ACM, 2011.

[169] Abdou Youssef. Cartesian product networks. In *ICPP (1)*, pages 684–685, 1991.

[170] Abdou Youssef. Design and analysis of product networks. In *Proceedings of the Fifth Symposium on the Frontiers of Massively Parallel Computation (Frontiers'95)*, pages 521–528, Washington, DC, USA, 1995. IEEE Computer Society.

[171] Chong Zhang, Zhangang Lin, and Zuoquan Lin. Variable neighborhood search with permutation distance for qap. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 81–88. Springer, 2005.

[172] Dawid Zydek and Henry Selvaraj. Fast and efficient processor allocation algorithm for torus-based chip multiprocessors. *Comput. Electr. Eng.*, 37:91–105, January 2011.