Scheduling and Dynamic Provisioning for Energy Proportional Heterogeneous Infrastructures

Violaine Villebonnet

Numéro National de Thèse : 2016LYSEN057

THÈSE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée par

l'École Normale Supérieure de Lyon

École Doctorale N° 512
en Informatique et Mathématiques de Lyon

Spécialité de doctorat :
Informatique

Soutenue publiquement le 6 décembre 2016, par :

Violaine VILLEBONNET

Scheduling and Dynamic Provisioning for Energy Proportional Heterogeneous Infrastructures

Ordonnancement et Allocation Dynamique de Ressources pour des Infrastructures Hétérogènes à Consommation Énergétique Proportionnelle

Devant le jury composé de :

Helen KARATZA	Professeure - Université de Thessalonique, Grèce	Rapportrice
Domenico TALIA	Professeur - Université de Calabre, Italie	Rapporteur
Christine MORIN	Directrice de Recherches - Inria Rennes	Examinatrice
Veronika REHN-SONIGO	Maîtresse de Conférences - IUT Besançon-Vesoul	Examinatrice
Laurent LEFEVRE	Chargé de Recherches - Inria ENS Lyon	Directeur
Jean-Marc PIERSON	Professeur - Université Toulouse III	Co-Directeur
Georges DA COSTA	Maître de Conférences - Université Toulouse III	Invité
Patricia STOLF	Maîtresse de Conférences - IUT Blagnac	Invitée

Acknowledgments / Remerciements

I would like to thank the members of the jury: Helen Karatza and Domenico Talia for reviewing my manuscript, Christine Morin and Veronika Rehn-Sonigo for having accepted to evaluate my work, and I thank them all especially for their presence at the defense of my thesis.

I will continue the acknowledgments in French...

J'estime avoir eu beaucoup de chance pendant ma thèse, et je voudrais remercier tous ceux qui y ont contribué :

En premier lieu, je remercie évidemment mes directeurs de thèse : Laurent et Jean-Marc, et mes co-encadrants de l'ombre : Georges et Patricia. Je sais que mes mails du vendredi étaient souvent trop longs... et je vous remercie d'avoir été si enthousiastes pendant les *skype* du lundi, c'était une source importante de motivation. J'ai vraiment été bien soutenue durant ma thèse, je suis donc ravie de tous vous avoir à ma soutenance.

J'ai eu la chance de me déplacer pour des conférences, événements et collaborations en France et dans le monde. J'ai pu faire de beaux voyages : Italie, Australie, Îles Canaries, Taïwan, et bientôt Chine. Les missions en France : Rennes, Nice, Grenoble, Lille, bien que moins exotiques, me laissent aussi de très bons souvenirs. J'en suis très reconnaissante, et je remercie toutes les personnes, et structures, qui ont rendu cela possible.

Au quotidien, les collègues de bureau sont importants, c'est pourquoi je remercie chaleureusement Leandro, Hongyang, Pedro et Radu. Je décerne à Pedro la Palme d'Or du co-bureau, et je me sens à la fois coupable, et très heureuse, de t'avoir entraîné avec moi dans le bureau 376N pendant un an et demi.

Je remercie bien sûr l'ensemble de mes collègues, des équipes Sepia à Toulouse et Avalon à Lyon, pour tous les moments passés ensemble. Je ne me lance pas dans une liste, qui serait forcément non exhaustive, toutefois, j'ai une pensée pour ceux avec qui j'ai partagé l'expérience de l'enseignement, et j'adresse un merci spécial à Hélène, ma seulE collèguE d'équipe lyonnaise.

Je m'estime chanceuse que beaucoup de mes collègues soient devenus des amis, et que l'un d'entre eux ait même acquis un statut un peu plus particulier. Je n'oublie pas pour autant mes amis de toujours, que je remercie infiniment de ne pas m'oublier non plus, et qui sont toujours partants pour m'héberger, me rendre visite, partir en voyage, ou discuter par chat interposé.

Et pour finir je remercie ma famille, toujours là pour moi où que je sois.

Abstract

The increasing number of data centers raises serious concerns regarding their energy consumption. These infrastructures are often over-provisioned and contain servers that are not fully utilized. The problem is that inactive servers can consume as high as 50% of their peak power consumption.

This thesis proposes a novel approach for building data centers so that their energy consumption is proportional to the actual load. We propose an original infrastructure named BML for "Big, Medium, Little", composed of heterogeneous computing resources: from low power processors to classical servers. The idea is to take advantage of their different characteristics in terms of energy consumption, performance, and switch on reactivity to adjust the composition of the infrastructure according to the load evolutions. We define a generic methodology to compute the most energy proportional combinations of machines based on hardware profiling data.

We focus on web applications whose load varies over time and design a scheduler that dynamically reconfigures the infrastructure, with application migrations and machines switch on and off, to minimize the infrastructure energy consumption according to the current application requirements.

We have developed two different dynamic provisioning algorithms which take into account the time and energy overheads of the different reconfiguration actions in the decision process. We demonstrate through simulations based on experimentally acquired hardware profiles that we achieve important energy savings compared to classical data center infrastructures and management.

Résumé

La consommation énergétique des centres de calculs et de données, aussi appelés « data centers », représentait 2% de la consommation mondiale d'électricité en 2012. Leur nombre est en augmentation et suit l'évolution croissante des objets connectés, services, applications, et des données collectées. Ces infrastructures, très consommatrices en énergie, sont souvent sur-dimensionnées et les serveurs en permanence allumés. Quand la charge de travail est faible, l'électricité consommée par les serveurs inutilisés est gaspillée, et un serveur inactif peut consommer jusqu'à la moitié de sa consommation maximale. Cette thèse s'attaque à ce problème en concevant un data center ayant une consommation énergétique proportionnelle à sa charge.

Nous proposons un data center hétérogène, nommé BML pour « Big, Medium, Little », composé de plusieurs types de machines : des processeurs très basse consommation et des serveurs classiques. L'idée est de profiter de leurs différentes caractéristiques de performance, consommation, et réactivité d'allumage, pour adapter dynamiquement la composition de l'infrastructure aux évolutions de charge. Nous décrivons une méthode générique pour calculer les combinaisons de machines les plus énergétiquement efficaces à partir de données de profilage de performance et d'énergie acquis expérimentalement considérant une application cible, ayant une charge variable au cours du temps, dans notre cas un serveur web.

Nous avons développé deux algorithmes prenant des décisions de reconfiguration de l'infrastructure et de placement des instances de l'application en fonction de la charge future. Les différentes temporalités des actions de reconfiguration ainsi que leur coûts énergétiques sont pris en compte dans le processus de décision. Nous montrons par simulations que nous atteignons une consommation proportionnelle à la charge, et faisons d'importantes économies d'énergie par rapport aux gestions classiques des data centers.

Contents

List of Figures

List of Tables

List of Algorithms

– Chapter 1 –

Introduction

In the last decade, our society has witnessed an explosion of the use of on-line services and applications in the daily life. This evolution goes with the ubiquity of connected mobile devices: smartphones, tablets, watches, cars, and so on. Social networks and medias are now an inherent part of our habits. The web site `www.internetlivestats.com` lists internet activity happening in one second. At the time of the writing, it reports that in one second, 2780 Tweets have been sent, 732 Instagram photos have been uploaded, *127177* YouTube videos have been viewed and 55 323 Google searches have been done.

What users sometimes do not realize, is that at the other end of these applications, there are data centers – huge facilities filled with lots of computers, that are processing, computing, storing, retrieving and sending data to make applications available 24/7 from (almost) anywhere on the planet – which consume a large amount of energy.

A recent report [53] estimates that data centers in the United States consumed 70 terawatt-hours (TWh) in 2014, accounting for about 1.8% of total U.S. electricity consumption. It represents a 4% increase from 2010 to 2014, which is a large shift from the estimated 24% increase for the period 2005-2010. And the energy consumption of data centers is expected to continue increasing in the future. This same report projects that U.S data centers will consume approximately 73 TWh in 2020.

Apart from being an important financial limitation for data center's operators, these high amounts of electricity can also be seen as CO_2 emissions, participating in the greenhouse effect. According to several weather agencies, 2015 has been the warmest year on records. Before that, 2014 was the champion, and now 2016 is on track to surpass the record as several months of this year are already the hottest observed.

The consciousness starts to reach a worldwide level. During the 2015 United Nations Climate Change Conference (COP21) in Paris, nearly 200 countries took part in negotiations and agreed on tackling climate change. The agreement contains measures to drastically reduce greenhouse gas emissions and curb global warming to less than 2 °C by the end of the century. It is urgent to listen to the messages from our planet and be conscious of the inevitable issue we are facing.

In this thesis, we propose to make these infrastructures more efficient by improving their ability to adapt their energy consumption to the workload evolutions over time.

1.1 Large-Scale Distributed Systems

A distributed system is defined as a set of several computers, interconnected with a network, that coordinate their actions by exchanging messages in order to achieve a common goal. This goal differs depending on the purpose served by the distributed system. It can be to provide websites, applications or services such as mail, instant messaging or file hosting. These systems can also be meant to complete complex computations for applications requiring an important computing power, such as weather forecasting or DNA sequencing. This is usually referred to as *High Performance Computing (HPC)*.

Concretely, these computers, commonly referred to as *servers* or *nodes*, are stored vertically into cabinets called racks, themselves organized horizontally into ails inside a room, a floor, or a dedicated warehouse. Such facilities are known as *data centers*, or *supercomputers* in the HPC case.

Distributed systems dedicated to computing can be classified into different categories. The order used for their description also follows the chronological order of their appearance in research.

- **Cluster** defines a distributed system composed of tightly connected nodes, generally of homogeneous hardware, situated in the same location, which can be considered as a single computing resource. When private, a cluster is in general managed and used exclusively by one company.
- **Grid** qualifies a set of interconnected clusters, usually geographically distributed and possibly composed of heterogeneous servers. A software layer is necessary to homogenize the infrastructure in order to use it in a transparent manner.
- **Cloud computing** refers to a shared infrastructure that can be used on-demand by any customers, who will pay the cloud provider according to their usage. Resources are considered elastic as they are dynamically provisioned according to the users' needs. Cloud computing is divided in three main types of services :
  - *Software-as-a-Service (SaaS)*, denotes software and applications made available via internet browsers. Users have access to the services from anywhere, usually after having created an account, and without having to handle software updates or licenses.
    Examples of such services are: *Google Drive* for file storage and synchronized document editing, and *Flickr* for to image and video hosting and sharing.
  - *Platform-as-a-Service (PaaS)*, allows the users to build their own applications and deploy them on the provided platform. Users can focus on the development of their applications and launch them immediately when their are ready without having to build an managed the infrastructure to host them.
    Google App Engine is an example of PaaS. Applications are hosted in data centers managed by Google, where the resource scaling is automatically done.

- *Infrastructure-as-a-Service (IaaS)* gives users direct access to machines they can fully set up, configure, manage and use according to their needs. Users can rent bare-metal servers but most of the time, cloud providers offer different configurations of *virtual machines (VM)*. *Virtualization* is a key technology which has enabled the expansion of cloud computing. It allows several independent operating systems to coexist on a single physical machine. Multiple virtual machines can be hosted on the same server while each of them being totally isolated from the others.
  Amazon Elastic Compute Cloud (EC2) is arguably the most famous cloud provider, while *OVH* is a French server and cloud provider.

1.2 Motivations for Energy Savings in Data Centers

The electricity consumed by a server is used to power its many components and perform computations, but a part of it is transformed and rejected in the form of heat. Unfortunately, a server is not able to bear high heats and its probability of failure increases with the temperature. Even if failures are most frequent at high temperatures, operators prefer to adopt conservative approach to increase hardware reliability as much as possible [20]. Inside a data center room, the high density of servers in racks worsen the heat dissipation issue. That is why data center facilities need an important cooling system, with air conditioning units and fans, to extract the generated heat and keep the room at a reasonable temperature. The consumption of the cooling system generally accounts for 30 to 50% of the total electricity consumed by the data center [19].

Indeed, in a data center, not all the consumed energy goes to computing. This is highlighted by the *Power Usage Effectiveness (PUE)* metric, promoted by the Green Grid consortium in 2007 [27]. It is defined as a ratio of the total energy consumed by the whole data center over the effective energy consumed only by computing servers. This metric reveals all the overhead electricity consumed by cooling infrastructures, power supplies, lights, and so on. The ideal PUE is 1.0 as it means that all of the energy is used for computing. While a recent survey [58] estimates that the average PUE of respondents' largest data centers is around 1.7, *Google* advertises an average PUE of 1.12 for all their data centers in quarter 2 of 2016 [25]. Despite all the controversy about how companies compute and sometimes use this metric only for publicity purpose, the PUE can also be criticized because it does not show the real energy efficiency of the infrastructure.

Most data centers are over-provisioned to be ready to sustain unexpected load peaks, but they sometimes also contain numerous servers that are not fully utilized. An Uptime Institute survey [58] suggests that close to 30% of servers in U.S. enterprises' data centers are *comatose*, meaning that they are consuming power but not doing any useful work. This comes from the poor management of the infrastructure once it is operational. Aging or partly faulty servers remain in the data center, consuming energy, instead of being decommissioned. In such situations, the energy consumed by these inactive servers is effectively consumed by IT equipment, accounting positively for to the PUE metric, but is in really a bad energy utilization.

1.3 Focus on Energy Proportionality

In addition, people usually only focus on the maximum energy consumption of a data center, but not enough on the day to day consumption which varies a lot. Having a good energy efficiency at full load is important, but also when the load is low, and this aspect is sometimes forgotten. This issue has been exposed by Luiz Andre Barroso and Urs Holzle in 2007 [9]. They have observed more than five thousand servers from a Google data center, and noticed that they are mostly utilized at a load between 10 and 50%. This means they are rarely completely unused, and therefore in a state where they could be shut down, and also rarely at full performance, where they are the most energy efficient.

The main problem is that when a server is idle, i.e., powered on but without activity, its energy consumption is already significant. Some idle servers can consume as high as 50% of their peak power consumption [9]. This amount of energy consumed when a machine is idle is called the static consumption. With our researches we want to find how the static costs of a data center can be reduced as much as possible in order to have an energy consumption fully dynamic and only dependent on its utilization. The objective of this thesis is to answer the following question: An infrastructure whose energy consumption is proportional to its actual load is it achievable?

1.4 Contributions

In this thesis, we propose to reach energy proportionality with an original data center design composed of heterogeneous computing resources. We name this infrastructure *BML* for *"Big, Medium, Little"* to highlight the differences of characteristics of the chosen hardware. The heterogeneity in our infrastructure is considered at the level of the architecture. We propose to gather different types of architectures, as opposite as x86 and ARM, inside the same data center, to benefit from their specific performance and energy consumption characteristics.

To achieve energy proportionality, the objective is to always use the least energy consuming hardware, or combination of hardware, that meets the current needs of the running applications. This concept of adaptability is particularly relevant when facing applications whose workloads vary over time. In our infrastructure, we consider that, at any time, only the most appropriate set of hardware for the current load is powered on. The unused nodes are switched off, or put in a suspend or hibernate mode, allowing us to reduce static costs. Around this heterogeneous infrastructure we propose a scheduler that handles reconfiguration decisions, such as dynamic application migrations and management of computing resources by switch on and off actions.

The relevance of reconfiguration decisions relies on the preliminary phase of machine profiling that allows to know perfectly the energy and performance behaviors of the different types of machine for the target application. This enables to compute the most energy efficient combinations of machines for different application performance levels. Based on the knowledge of the application workload profile, the scheduler enforces infrastructure reconfigurations to reduce energy consumption while respecting QoS constraints of the application. The design of our BML infrastructure with its general framework, as well as our first experimental results have been published in the proceedings of *BDCloud 2014*

[C4], and in *Parallel Processing Letters 2015* journal [J1] in an extended version.

We propose two versions of scheduling algorithms: the first one called *Ideal BML*, only considers the machine combinations computed in the preliminary profiling phase. It has been the subject of two publications, a short paper at *Cluster 2016* conference [C1], and a full paper at *ICPADS 2016* [C3]. The second algorithm named *Multi-Terms*, allows more reconfiguration possibilities and take care of the different temporalities of these actions. It is explained in details in a paper published at *Sbac-Pad 2016* conference [C2]. We have made experimental evaluations which demonstrate that we are saving substantial amounts of energy compared to classical data center managements as we drastically reduce the static energy costs thanks to dynamic reconfigurations of heterogeneous computing resources.

1.5 Structure of the Manuscript

The structure of the manuscript is the following: Chapter 2 provides an overview of the researches concerning energy proportional computing. This state of the art study leads us to heterogeneous computing resources as a key to reach energy proportionality. Hence we develop in Chapter 3 the technological challenges brought by heterogeneity inside a data center and justify our choices. We detail in Chapter 4 our framework *"Big, Medium, Little"* and its specifications. We especially focus on the step by step process for building the heterogeneous combinations of computing resources. Chapter 5 presents our first scheduler version named *Ideal BML algorithm* with its evaluation. Chapter 6 describes the second version of the scheduler called *Multi-Terms algorithm.* We evaluate it considering the same scenarios to highlight the differences compared to the previous version. Chapter 7 contains discussions about the relevance of our proposition, the hardware choices, the possible improvements, and how we picture the evolutions of this approach in the future. Finally, Chapter 8 concludes our work by summarizing our achievements and gives insights for actual implementation and directions for future works.

– Chapter 2 –

Energy Proportional Computing: State of the Art

This chapter presents the state of the art of researches concerning energy proportional computing. Although this term has been defined relatively recently, many works trying to enhance energy efficiency of servers and data centers can be seen within the energy proportionality goal.

2.1 Energy Proportionality

2.1.1 Definitions and Goals

Andre Barroso and Urs Hölzle, two engineers at Google, published a paper in 2007 entitled: "The Case for Energy-Proportional Computing" [9]. In this work, they exposed the energy inefficiency of servers with measurements done inside data centers of their own company. Their two key findings are: (i) Servers are utilized on average between 10% and 50% of their maximum capacity; (ii) When idle, a server can consume up to 50% of its peak energy consumption.

After making these observations, their conclusion is that energy proportional servers would bring large energy savings. Therefore, they urged hardware designers to enhance the energy proportionality of servers and data center operators to improve the resource management in their infrastructures. Themselves did not give solutions to achieve these goals but they opened the research field of **Energy Proportional Computing**.

In mathematics, two variables are said *proportional* if a change in one is always accompanied by a change in the other, and if the changes are always related by use of a constant multiplier. Applying this definition to our problem, we want the energy consumption and the utilization of a server to be *proportional*. By utilization we mean the load of the server, which is characterized by the amount of work performed. Consequently, a perfectly proportional server is a machine consuming 0 when idle and having an energy consumption always proportional to the amount of work produced until its maximum

capacity. The constant multiplier is in our case the *energy efficiency* coefficient, defined as utilization level divided by the corresponding energy consumption.

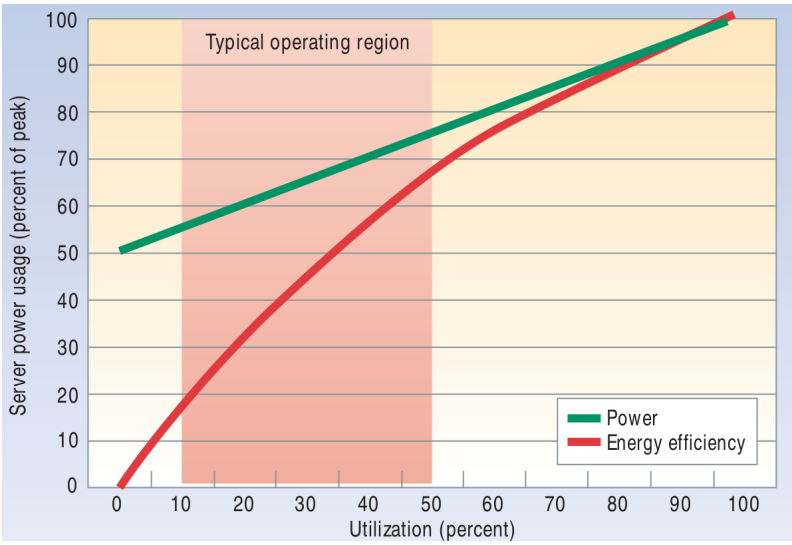


Figure 2.1: Typical power consumption of a server from 0 to 100% utilization (from [9])

On Figure 2.1, extracted from [9], the upper green curve represents the typical energy consumption of a 2007 server from 0 to 100% utilization, while the lower red curve depicts its energy efficiency. As the idle consumption of this server accounts for 50% of its peak consumption, its energy efficiency is very poor for the lower range of utilization. Unfortunately, the typical utilization range Barroso and Hölzle have observed was between 10% and 50%, which corresponds to a low energy efficiency. That is why an energy proportional server which has a constant energy efficiency regarding its utilization would enable important energy savings.

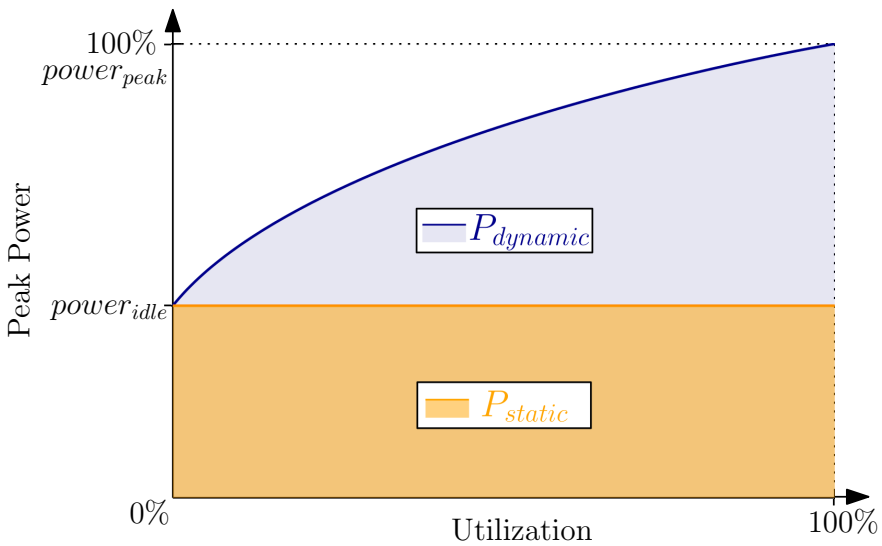


Figure 2.2: Dynamic and static parts for a typical server power consumption

Figure 2.2 pictures another point of view of the problem that is static versus dynamic parts of power consumption. Indeed, the area from zero to idle consumption is considered as the static consumption part. This fixed amount of power consumption no matter the utilization is the one that prevents a server from being energy proportional. Our goal of energy proportionality can be reached only if static costs disappear and the power consumption of a server is only composed of one dynamic part.

2.1.2 Metrics

As it is elementary to know how to measure and quantify a phenomenon in order to understand it fully, several works defined metrics to express energy proportionality characteristics of servers.

Ryckbosch et al. [50] proposed a metric which is computed using the area between the actual power consumption curve of the server, and the ideal proportional consumption curve, i.e., starting at zero in idle state and being proportional towards peak power and performance, as represented on Figure 2.3. This metric, named *EP* for *Energy Proportionality*, is defined like this:

$$EP = 1 - \frac{area_{actual} - area_{ideal}}{area_{ideal}} \tag{2.1}$$

It quantifies in a global way how the server is close to perfect energy proportionality: an EP of 1 results from perfect proportionality while 0 characterizes a server consuming a constant amount of power irrespective of its utilization level. The server pictured in Figure 2.1, whose idle power consumption is 50% of its peak power, has an EP of 0.5. Although this metric gives a good hint about the energy proportionality of a machine, it can be reductive as two machines can have the same EP score while having two different power consumption curves.

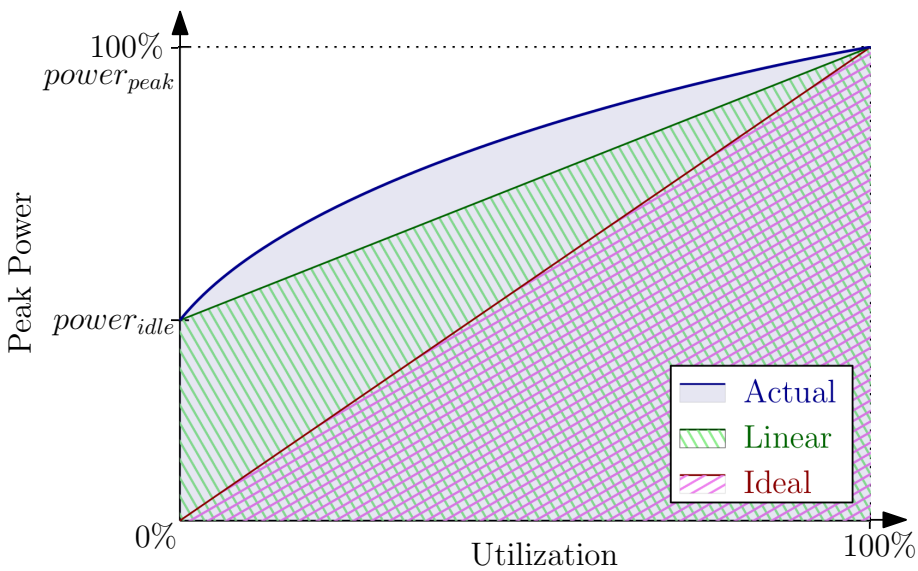


Figure 2.3: Actual, linear and ideal proportional power consumption curves

Hence, following works [60, 62] refined this work by proposing two metrics to quantify the two different aspects that portray energy proportionality.

The first one is the dynamic power range i.e., the difference between the idle and the peak power consumption. Varsamopoulos et al. [60] defined the *IPR* metric for *Ideal to Peak Ratio*. It is computed as the ratio of the power consumed when idle over the power consumed at full utilization:

$$IPR = \frac{power_{idle}}{power_{peak}} \tag{2.2}$$

A lower IPR denotes a larger range of dynamic consumption and thus a more energy proportional server. In the same idea, Wong et al. [62] proposed the *DR* metric, standing for *Dynamic Range*. It also uses idle and peak power consumption values, but is computed as follows:

$$DR = \frac{power_{peak} - power_{idle}}{power_{peak}} \tag{2.3}$$

An energy proportional system would have a dynamic power range DR of 1.

The second aspect that needs to be evaluated is the linearity of the consumption. For that aim, it needs the definition of a hypothetical linear power curves which starts at same idle consumption and ends at same peak consumption as the considered system, but is linear between these two points. Varsamopoulos et al. and Wong et al. defined again two quite similar metrics to characterize linearity. They are respectively *LDR* for *Linear Deviation Ratio* in [60] and *LD* for *Linear Deviation* in [62], computed as such:

$$LDR = \max_{i}^{|.|} \left(\frac{power^{i}_{actual}}{power^{i}_{linear}} - 1 \right) \tag{2.4}$$

$$LD = \frac{area_{actual}}{area_{linear}} - 1 \tag{2.5}$$

The LDR and LD metrics have another advantage compared to the EP metric because the result keeps the sign of the maximum deviation from the linear curve. This means that depending wether the value is positive or negative, we know if the server's power curve is over or under the theoretical straight line. A LD or LDR value of 0 corresponds to a perfectly proportional, or linear, power curve. Negative values indicate under-linearity, which sometimes implies that a server consumes less than the proportional goal. It is better than over-linearity concerning energy savings, but it can signify that a server is not the most energy efficient at full utilization, which can be difficult to take into account in scheduling algorithms.

We present these metrics for actual hardware, allowing to highlight and discuss their differences with actual examples in section 4.4.5.

2.1.3 Recent Evolutions

Once metrics have been defined, these works [60, 50, 62] studied existing servers to evaluate how energy proportionality has evolved over different successive generations. All their analyzes are based on the *SPECpower_ssj2008* benchmark [26] whose results are published by the *Standard Performance Evaluation Corporation (SPEC)* [1]. The results contain entries from several hardware vendors since 2007. Each entry contains idle power consumption as well as power measurements corresponding to ten different utilization levels, thus all the data needed to compute energy proportionality metrics.

The paper of Hsu et al. [30], published in 2015, is the most recent study. It covers all SPECpower results from 2007 to 2014, and evaluates the evolution of the five metrics previously described. It is clearly observable that servers have been increasingly energy proportional over these seven years. In 2007, servers' EP was between 0.2 and 0.4 while last entries from 2014 are around 0.7 and 0.8. As DR and IPR metrics are quite similar, they observed the same positive evolution. In contrast, the progression of the metrics LD and LDR which characterize linearity is unclear. Varsamopoulos et al. [60] provided an interesting study showing that as IPR has improved over time, meaning that servers have greater dynamic power range, LDR is getting worse. And more specifically, two trends are noticeable: servers are mostly deviating from linearity positively, i.e., above the linear curve, but few servers are negatively diverging from linearity.

As a conclusion, despite an improved energy proportionality, current servers' consumption is still not perfectly proportional to their load. Moreover, hardware manufacturers have put effort on increasing dynamic power range, but at the cost of a degraded linearity. Therefore there is still room for improvements regarding energy proportional computing. In the following sections, we describe the many different approaches that aim at getting closer to this objective with existing non proportional hardware.

2.2 Energy Efficiency of Homogeneous Clusters

2.2.1 At Software Level

Efforts can be made during software development in order to save energy during its execution. Indeed, a software program that is not coded with the awareness of the underlying hardware on which it will run, can lead to substantial energy wastes. Energy-aware programming practices are often referred to as *Green Coding* or *Green Programming*.

The concept of *Race to Idle* is quite simple: increasing the software performance, leading to minimizing its running time, saves also energy because the faster the execution is completed, the earlier the computer is back to idle and can be put in an energy saving mode. Thus, a first green programming practice consists in developing efficient algorithms using for example multithreading and vectorization for parallel executions. Sabharwal et al. [51] described such recommendations and techniques to build energy efficient software, and focused on reducing energy consumed by idle software instances. Authors have studied different idle applications, e.g., a media player which is open but not playing anything, or an open internet browser not displaying any site, and measured

[1] SPECpower_ssj2008 results are available at `https://www.spec.org/power_ssj2008/results/`.

their power consumption. They advise programmers to limit as much as possible waiting loops and active polling that frequently wake up the CPU and result in wasting electricity.

In the same lines as energy proportional computing, Saxe [52] claimed that the amount of resources consumed should be in direct relation with the amount of work actually done by the application. He detailed some key principles to reach this objective such as reducing unnecessary computation, using event-triggered actions instead of active waiting. He also recommends to perform input and output accesses in batches as well as avoiding memory leaks and freeing no longer needed memory.

More optimizations can also be done during the code compiling process. This is what Fakhar et al. [23] proposed with their green compiler. It aims at making code more energy efficient by applying several techniques which are cache skipping, using register operands, clustering and reordering instructions, loop optimization and so on.

While these green programming practices are important to consider when possible, sometimes users are not authorized to modify the code of the applications they are working with. Even if they can, it may be fastidious to modify a large number of existing lines of code, and the impact on server's energy consumption may not always be worth the effort. That is why it is necessary to couple green coding with energy savings actions at runtime.

2.2.2 At Server Level

At the server level, two main approaches can be distinguished: *shutdown*, which focuses on improving switches towards energy saving modes and back, and *slowdown*, which consists in making the dynamic consumption more related to the actual work performed.

Static costs are tackled by *shutdown* approaches, which propose to put the processor in some low power modes when it is not active. The *Advanced Configuration and Power Interface (ACPI)* specification [2] defines different *C-states*: *C0*, the operating state during which P-States can be chosen, and *C1* to *Cn*, the idle states during which some processor's components are disabled to save energy. The higher the number of *Cn*, the less energy the processor consumes and the longer it takes to make it active again. ACPI also defines several sleep states *S-states*, during which most of the system is powered off, except the network card that stays active to support *Wake-on-LAN*. S-States include *Suspend-to-RAM*, commonly called *Standby* mode, and *Suspend-to-Disk*, also referred to as *Hibernate* mode.

As transitions between idle or sleep states and active state are quite long, idle periods should be enough long to benefit from these techniques and save energy. Indeed, in [3], Amur et al. proposed *IdlePower*, a solution that batches timing interrupts sent to virtual machines in order to maximize the duration of idle periods and consequently the amount of time spent in deep C-states. However, they also noticed that application performance is degraded when idle management is too aggressive because deep C-state incurs cache losses. PowerNap [41] models how quickly servers would have to switch to and back from sleep states to achieve energy proportionality for several workloads. Meisner et al. conclude that a transition time of 10ms or less is necessary to achieve significant energy savings. Unfortunately, today's server are very far from this objective.

On another level, *slowdown* approaches concern only the dynamic part of the power consumption. *DVFS*, standing for *Dynamic Voltage and Frequency Scaling* is a well known technique which allows to reduce power consumption of computing systems by adapting both their voltage and frequency according to the processed workloads. By reducing the operating frequency of a processor, its voltage can also be lowered, which leads to a decreased consumption of electricity. Several commercial processors support DVFS technology, e.g. *PowerNow!* and *Cool'n'Quiet* from AMD and *SpeedStep* from Intel. Each of these processors can operate at different *performance states* with predefined frequencies, which are called *P-states*. On Linux operating systems, *CPUfreq* enables to change the frequency according to the chosen governor policy among these several implementations: *performance*, which always selects the highest frequency; *user-space*, allowing the user to set manually the frequency; *on-demand*, which automatically adjusts the frequency; *conservative*, which also adjusts automatically the frequency but increasing it progressively; and *powersave*, which always chooses the lowest frequency.

The limitation of DVFS is that it reduces the performance and consequently lengthens the execution time. Therefore, it is necessary to determine the most appropriate frequency scaling in order not to increase the overall energy consumption. Dargie showed in [18], where he studied the impacts of DVFS in a multimedia server use case, that this technique enables important gains for I/O bound workload, but it is generally unfavorable when the CPU is utilized at more than 40% on average. Moreover, this work highlighted that it is necessary to know the workload to be able to correctly choose the most appropriate governor policy. As the transitions between frequencies are not instantaneous, on-demand policy may not be relevant for very bursty workload.

Another server power management technique has been developed by Intel in their Sandy Bridge processors. It is named *RAPL*, standing for *Running Average Power Limit*. Via this mechanism, a user can specify a power consumption threshold that the processor will not exceed during a given period. The system then automatically regulates its behavior to keep its consumption under the limit. The energy savings achieved by RAPL have been evaluated facing different use-case applications: Subramaniam and Feng et al. focused on data stores in [55] and Lo et al. on latency critical workloads in [35]. This power capping technology offers better results than DVFS because it can be controlled at a finer grain.

All these works at server level are focused mainly on CPU because it is almost always the most consuming component of a server. According to Barroso et al. in their study considering data center as a computer [8], CPUs account for approximately 42% of the energy consumption of a classical data center in 2012. However, other parts are responsible for the disproportionality of the energy consumed by a server and some improvements can also be done in those fields. Memory and storage systems are also responsible for a substantial amount of a server's energy consumption. Malladi et al. [39] wanted to counteract the energy inefficiency of *Dynamic Random Access Memory (DRAM)* by using in servers memory systems that are originally designed for mobile platforms. As storage systems are concerned, slowdown and shutdown methods also exist, consisting in reducing the spinning speed of the disk for instance. Kim and Rotem [31] proposed to reduce the energy consumed by storage systems by exploiting the data replication usually done in data centers. As data is replicated, it is easier to get the data from an already spinning disk instead of waking up a disk that is in sleep mode.

2.2.3 At Cluster Level

So far we have seen methods to improve the energy proportionality of individual servers, but we will now change perspective and look at the energy proportionality at the cluster level. Of course if a cluster is composed of energy proportional servers, there is no doubt that the cluster will itself have a perfect proportional energy consumption. However, as such servers do not exist yet, there are actions that can be done at the cluster level to reach energy proportionality even if it is composed of non proportional servers. The whole idea resides in powering on only the machines that are needed to meet the current demand. All unused servers are kept powered off and are switched on only when they become necessary. This allows to enhance the energy proportionality of a cluster as its composition is dynamically adapted to the load fluctuations. Nevertheless, this idea is not as simple as it seems because switching off and on a server takes time and consumes extra power. Hence these actions must be well decided to actually save energy.

Many works have been conducted exploring this idea, and several have named this approach *cluster resizing*, or *Dynamic Provisioning*. In one of the earliest works, in 2001, Pinheiro et al. have proved with simple experiments the potential energy savings of dynamic provisioning for a small cluster of 8 servers serving web applications [47]. The minimum of physical nodes is powered on while the load is balanced among them.

Later, with the apparition of virtualization technologies, the concept of *Consolidation*, which consists in colocating several virtual machines on the same physical server in order to minimize the quantity of utilized resources, has allowed to broaden the range of applications. It was made possible thanks to the mechanism of *Live Migration* [14], which is used to dynamically move virtual machines from a physical server to another without much impacting the performances of the applications running inside. When several virtual machines have little load, they can be hosted on the same server, and as soon as one of them needs more resources, a new physical machine can be provisioned and this virtual machine migrated to it.

Most consolidation approaches are based on *heuristics* algorithms, whose goal is to solve variants of the bin packing problem, where virtual machines must be packed on physical servers while respecting their resources requirements e.g., CPU or memory utilizations. More original alternatives have been proposed, for instance Feller et al. [24] have developed a nature-inspired algorithm based on the Ant-Colony meta-heuristic to compute the workload placement dynamically according to the current load. Hermenier et al. used constraint programming in [29], allowing them to take reconfiguration overheads into account. Live migration being a costly process in terms of time and energy, it is necessary to use it wisely. Balouek-Thomert et al. [7] proposed to perform dynamic server provisioning at the middleware level, while taking into account energy-related events and user preferences concerning performance and energy ratio. A more complex problem has been approached by Ardagna et al. as they focused on web services placement across multiple geographically distributed cloud sites [4]. Consequently, they have defined two types of actions: resource provisioning at mid-long term across different cloud sites, and virtual machine provisioning inside a same site at short term.

Tolia et al. [57] have widen their study of cluster level energy proportionality as they also measure the energy consumed by cooling systems. Indeed, they proposed a predictive approach to control the fans. They showed that both servers and cooling systems can get

closer to energy proportionality is they are well managed.

Finally, Wong and Annavaramn [61] have evaluated the influence of server level energy proportionality on cluster level. As servers are becoming more and more proportional, they wanted to know if highly energy proportional servers would exempt more energy managements at cluster level. The conclusion is that even if server level proportionality has a direct impact on cluster level, since idle servers' consumption is not close to zero and on/off switches are still lasting too long, there is still a need for more energy efficient managements at cluster level.

The main limitation of these cluster-level actions resides in the homogeneity of computing resources that does not allow a fine grained provisioning. Servers have an important processing capacity and an important energy consumption, thus a decision of switching on or off a server has a big impact on the infrastructure. Therefore, some researchers have recently gained interest for low power processors and started to study if they can offer more energy efficient platforms.

2.2.4 Low Power Processors Clusters

With the emergence of mobile devices, hardware designers have been constrained to conceive low power processors to extend battery life. Moreover, they were also encouraged to build more and more powerful processors as many applications running on smartphones and tablets are highly resource intensive, e.g., games, video streaming, geolocalized applications. Mobile processors are mainly based on *ARM* architecture, which uses *Reduced Instruction Set Computing (RISC)*. Design of RISC processors requires less transistors than traditional *x86* processors based on *Complex Instruction Set Computing (CISC)*, that is why their power consumption and costs are reduced.

As mobile processors feature a good performance to power ratio, they seem to be good candidates for building energy efficient supercomputers. The European project called *Mont-Blanc*, started in 2011 [56], aims at designing a new type of architecture for HPC using energy efficient mobile processors. They were the first to propose a prototype HPC cluster built from ARM multicore processors, which they named *Tibidabo*. They performed many experiments with HPC workloads and concluded that applications that scale to a larger number of parallel nodes benefit from this platform having competitive performance and energy efficiency compared to standard servers [48]. However, they also found some limitations as their prototype was based on first generation of 32-bit ARM processors. Indeed, the new 64-bit ARMv8 architecture brought some improvements and in particular double-precision floating-point computing.

Other works have studied the use of low power processors for handling data center workloads like web servers and big data applications. Varghese et al. [59] showed that the popular *Raspberry Pi* is efficient for hosting static web servers while consuming less power than a standard server. Loghin et al. [36] also concluded that ARM processors offer interesting consumption-performance ratios for database query processing compared to an Intel Xeon processor. On the other hand, these studies show performance limitations, concluding that these low power equipments cannot compete with standard servers for more demanding workloads such as dynamic web servers or I/O intensive big data applications.

Several hardware vendors have started to propose server solutions based on ARM processors, and *HP Moonshot* [45] is one example. It exploits the concept of *Disaggregated Server* design where servers are disaggregated and resources, such as processors, memory and I/O ports are arranged in resource pools constructing processing pools, memory pools and I/O pools. The HP Moonshot system is a *software-defined server* as the company proposes to choose among different processors to optimize performances for specific workloads. Their ARM based solution is said to be optimal for web infrastructure.

As a conclusion, clusters composed of low power processors are very promising regarding energy proportionality, but are not necessarily suitable for all types of workloads. That is why heterogeneity seems to be the solution to take advantage of both low power processors and powerful servers.

2.3 Benefits of Heterogeneity in the Infrastructure

2.3.1 Inspirations from the Mobile World

To be even more efficient and further extend the battery life of mobile devices, heterogeneous multicore processors were introduced. ARM was the first to propose a heterogeneous architecture, which they called *ARM big.LITTLE*, [6] consisting in putting two different processors on the same board. The idea is to offer a processor with very low power consumption that delivers low-level performance, and a more powerful, and consequently more power consuming processor, to process more intensive tasks. Indeed, this technology takes advantage of the dynamic usage pattern of smartphones and tablets. These devices alternate between highly intensive tasks such as initial web page rendering and game physics calculation, and low processing intensity tasks such as reading a web page, waiting for user input, and light tasks like texting, e-mail and audio. ARM big.LITTLE has a total of eight cores, four cores for each type of processors: *"little"* is a Cortex-A7 and *"big"* is a Cortex-A15. The two processors are connected through a cache coherent interconnect which allow transparent and efficient data transfer between them.

When the processor first came out, its first implementation, termed *Clustered Switching*, only allowed to use one cluster at a time, either the big or the little one. Then the *In-kernel switching* approach proposed to pair each big core to a little one, resulting in a 4 core heterogeneous processor as only one core of each pair could be used at a time. To benefit from the full potential of big.LITTLE, the best model is *Global Task Scheduling (GTS)*, giving the ability to schedule tasks on all the cores at the same time. ARM has also developed a kernel space patch based on GTS that keeps track of load history as each thread runs, and uses this data to anticipate the performance needs of the thread next time it runs.

Samsung has implemented this architecture in their *Exynos Octa* boards. They can be found for example in *Samsung Galaxy S4* ans *S5* smartphones. Once *64-bit ARMv8* architecture was released, a new version of ARM big.LITTLE has been created with the two processors Cortex-A53 and Cortex-A57. This latter powered *Samsung Galaxy Note 4* and *Galaxy S6 series* among others.

Mediatek went even further by designing the first three-cluster processor, the *Helio*

x20. It is a deca-core processor composed of a dual-core ARM Cortex-A72 operating at a frequency up to 2.3 GHz, a quad-core ARM Cortex-A53 up to 1.85 GHz and another quad-core Cortex-A53 up to 1.4 GHz. It is embedded in the smartphone *Meizu MX6* available since July 2016. Mediatek has recently revealed its new *Helio x30* processor, using a 10nm manufacturing process, containing ARM Cortex-A73 and A35 cores, with huge performance capabilities and an improved power management.

2.3.2 Motivations for Heterogeneity in Data Centers

The trend of heterogeneous multi-core processors, in the same inspiration as the architecture of ARM big.LITTLE, has emerged in data centers. An example is the prototype *QuickIA* by *Intel*. This heterogeneous platform gathers one *Intel Atom* processor and one *Intel Xeon* processor, which are representative of two opposite architectures. The *Xeon* is a high-performance server-class architecture, while the *Atom* is a low-power micro-architecture targeted for mobile devices. Cong and Yuan [15] have developed a scheduling approach to take advantage of this heterogeneous platform. They used a regression model to estimate the energy consumption of each processor and take scheduling decisions accordingly. The drawback is that this approach requires the code to be instrumented to be able to predict the execution time and associated energy consumption.

Wong et al. have proposed *KnightShift* [62]. It consists in a motherboard containing a regular server processor, called *Primary Server*, and a low power processor, called the *Knight*. This latter is always powered on, and wakes up the primary server only in case of high load. In fact, they have defined several possible architectures and studied different processor candidates. As it is not simple to design custom motherboard, they have implemented *KnightShift* at the server level by using an Intel Xeon and an Intel Atom connected through their network interfaces.

In the European project *CoolEmAll* [33], that aimed at improving energy efficiency of data center by working on models, simulations and visualization tools, they used a heterogeneous server board called *RECS* for *Resource Efficient Computing System* developed by Christmann [12]. This server has a very high density as it consists in 18 single CPU modules of three different architecture types, from *Intel Atom* to *Intel i7*, and each node is equipped with thermal and energy sensors which facilitates monitoring.

The heterogeneity can also come from a computing architecture different from traditional processors. *Nvidia* introduced in 1999 the first *Graphics Processing Unit (GPU)*, which is a specialized electronic circuit dedicated to graphics rendering. A GPU is able to render images faster than a CPU because it has a parallel processing architecture that allows to process large blocks of data as multiple calculations are done at the same time. This characteristics made the GPU an interesting processing unit for *High Performance Computing (HPC)*. Indeed, if a computation is performed faster, it often implies that the energy consumed is decreased, but GPU represent also an energy overhead compared to CPU, that is why some researchers started to evaluate GPU's energy efficiency. Enos et al. [22] evaluated the energy improvement, quantified in performance-per-watt, for four different applications that have been ported to work on GPU. Authors concluded that although using GPU increases significantly power consumption, the resulting acceleration also incurs a reduction of the overall energy consumption.

The fact of using GPU to process computation that is traditionally handled by CPU, is called *General-Purpose Computing on Graphics Processing Units (GP-GPU)*. It is facilitated by the framework *Open Computing Language (OpenCL)*, which allows to write programs that will be executed across heterogeneous platforms, i.e., both on CPU and GPU. Thanks to this framework, it is no longer necessary to write two versions of the same code: one dedicated to run on GPU and one for CPU. *CUDA* is a parallel computing language by Nvidia and an implementation of the OpenCL standard. Ma et al. proposed *GreenGPU* [38], an energy management framework for GPU-CPU heterogeneous architectures. Their work consists in two steps: First, workloads are dynamically split and distributed to GPU and CPU based on their characteristics, so that both sides can finish approximately at the same time. Therefore, the energy consumed when staying idle and waiting for the slower side to finish is minimized. Second, the frequencies of CPU, GPU cores and memory are dynamically adjusted, based on their utilizations. They achieved important energy savings thanks to this advanced management.

Custom heterogeneous boards are not easy to build nor maintain, and GPU requires extra programming efforts. Heterogeneity can simply resides in the fact of using different types of servers in the same data center. This idea is not brand new as heterogeneity tends to naturally appear in data centers with time. Heath et al. [28] argued that maintenance can result in replacing some components with more powerful ones as computing needs were increasing drastically a decade ago, while cost versus performance ratios were decreasing. They advocated the use of an efficiency metric and the need to model the different types of nodes with respect to this metric. In their work, they used modeling and optimization to minimize the energy consumed per request in a web server use case.

Nathuji et al. [43] have also shown that resource heterogeneity can be exploited to improve energy efficiency. They were able to perform an efficient workload allocation with an analytical prediction model for computing power and performance of the different architectures with their respective power management capabilities.

While these two approaches consider successfully benefit from resource heterogeneity, they mostly consider it as a fatality. With the democratization of low power processors, several following works, including this thesis, support the idea of deliberately choosing a set of heterogeneous machines when building a data center. Chun et al. [13] made the case for *hybrid* data centers, meaning composed of heterogeneous machines, based on the observation that some applications have significantly different performance per watt on different platforms. Indeed, they have also selected the Intel Atom processor for its low power consumption, and Intel Xeon for high performance. They have performed experiments with various workloads such as web services, data mining, video conversion, and they highlighted that workload characterization is crucial to decide which processor will perform the most efficiently. The work of Krioukov et al. [32] also used these two same processors while focusing on web services. They have proposed several dynamic provisioning algorithm, computing the number of machines that should be on or off depending on the load evolution, but their evaluations are not based on real load traces.

Da Costa compared the energy efficiency of three different processors: Intel I7, Intel Atom and ARM Raspberry Pi, and evaluated the energy savings that could be achieved in a heterogeneous data center serving web services [16]. This work has only shown preliminary results for a best-case scenario, without taking into account energy and time overheads of machines switches on and off.

To wrap up, these heterogeneous solutions either rely on very specific and complex designs, or lack some aspects that prevent them to be implemented in reality. In this thesis, we are building a heterogeneous data center with existing hardware, and we develop scheduling algorithms which consider the different overheads and temporalities of the reconfiguration actions.

– Chapter 3 –

Feasibility of Highly Heterogeneous Data Centers

In this chapter we describe the preliminary studies and experiments concerning the virtualization technologies and how they can be used with heterogeneous computing resources. The results that we have drawn from these studies have guided us for the rest of our work.

3.1 Description of the Goals and Problems

Our main goal is to save energy by building an energy proportional data center with a set of heterogeneous machines to benefit from their different power consumption and performance characteristics. We consider a high heterogeneity because we want to mix ARM low power processors and x86 traditional servers inside a same infrastructure. The problem is that these two kinds of machines have different *Instructions Set Architecture (ISA)*. Consequently, it is necessary to evaluate the challenges brought by this heterogeneity, and to find the possible solutions to make these architectures *compatible* and both efficient.

Ideally, the objective would be to have the least amount of constraints concerning the applications types, and the fewest possible changes from typical data center management. That is why we have lead a study on the virtualization solutions available both on ARM and x86 architectures. Indeed, virtual machines or containers can embody any types of applications. The goal of this technical study is to determine if there exists a solution, whichever it is, that would enable executing applications across heterogeneous computing resources, and migrating application instances from one type of machine to another.

3.2 Study of Virtualization Solutions for ARM and x86

3.2.1 Virtual Machines

Virtualization allows multiple operating systems (OS) to execute concurrently on the same physical hardware, usually called the *host*. The operating system running on the

host is consequently named *host OS*. Each virtual machines (VM) can have its own operating system, also referred to as *guest OS*. The virtualization software, called *hypervisor*, is supposed to simulate hardware components so that each guest can concurrently access them. However, several levels of virtualization are defined. In *full virtualization*, everything is perfectly simulated in a way that all guest OS can run with no prior modifications. It is opposed to *paravirtualization*, where not all the hardware parts are simulated for performance improvements, and the guest can sometimes bypass the hypervisor and execute some tasks directly in the host domain.

We also distinguish two types of hypervisors. Type 1 is a separate software component that runs directly on the hardware and provides an abstraction to the virtual machines running on top of the hypervisor. A popular example of type 1 hypervisor is *Xen*. Type 2 runs an existing OS on the hardware and run both virtual machines and applications on top of this OS. Usually, this type of hypervisor slightly modifies the host OS to facilitate the execution of virtual machines. For instance, *KVM* is an example of type 2 hypervisor and is integrated in Linux and available as a loadable driver.

On x86 platforms, virtualization has already existed since the late 1990s, but it needed complex software techniques. In 2005 and 2006, both Intel and AMD introduced new processor extensions designed to ease the virtualization processes. This approach is known as *hardware-assisted virtualization* or *accelerated virtualization*.

ARM architecture differs a lot from x86 and it is not classically virtualizable [46]. Therefore, full virtualization of ARM platform has not been explored a lot, and suffered from poor performances. The development of ARM virtualization has really started when hardware virtualization extensions have been created in the latest ARMv7 and ARMv8 architectures. For instance, ARM Cortex-A15 was the first ARM processor with virtualization support. It has been released in 2012, and its first implementation was the Samsung 5 Dual board inside the *Samsung Chromebook Series 3*, and later in the *Google Nexus 10* tablet.

KVM/ARM was the first hypervisor to use ARM virtualization extensions to run unmodified guest operating systems [17]. As a result, KVM is available on all ARM platforms running a recent version of the Linux kernel as KVM/ARM has been merged in Linux kernel version 3.9, released in April 2013. *Xen on ARM* has been available since Linux kernel 3.7, but it necessitates some drivers development for the guest OS to be executed with paravirtualization.

3.2.2 Containers

As we are not tied to the virtual machine technology, but are only looking for a way to easily migrate applications between heterogeneous machines, we have studied the solution offered by containers. It is a technique of virtualization at the *operating system level*. In this case, the kernel of the host OS is shared and accessed by multiple isolated user-space instances. All these instances, called *containers* or sometimes *jails* in reference to their isolation property, look like real servers from the users point of view. This technique implies very little to no overhead and does not require any hardware virtualization extensions. However, it is not flexible as all containers instances can only be running the same operating system and even the same kernel as the host. Some proprietary solutions exist for Windows, but the majority of approaches are based on Linux. We have selected

two implementations that seem to have interesting characteristics, and studied if we can use them for our purpose.

OpenVZ, whose initial release was in 2005, uses a patched Linux kernel and works both on x86 and ARM. Its advantage is that it has a live migration feature. Unfortunately, this live migration technique has a longer downtime than virtual machine's migration, and is not operational on ARM architectures. *LXC*, also known as *Linux Containers*, initially released in 2008, uses the *cgroups* functionality of the Linux kernel which enables resources limiting and prioritizing concerning CPU, memory, I/O, etc. It is consequently available in the mainline Linux kernel since version 2.6.29. It has gained more attention in 2014 because a company called *Docker* was using it to propose a promising tool to package any application and run it on any Linux platform. It quickly became popular and since then its development has never stopped growing. LXC is also working both on x86 and ARM, but does not have any live migration feature. The *CRIU* project, standing for *Checkpoint/Restore In Userspace*, was launched to achieve the goal of checkpointing and restarting Linux containers, but it was not yet completed when we started this study, and is still under development now.

3.2.3 Synthesis and Preliminary Choices

All the characteristics of chosen virtualization and containerization solutions are synthesized in Table 3.1.

	Virtual machines		Containers	
	Xen	KVM	LXC	OpenVZ
On x86	yes	yes	since 2.6.29	patched kernel
On ARM	since 3.7	since 3.9	since 2.6.29	patched kernel
Live migration	yes	yes	not yet (CRIU project)	yes, but only x86
Guest OS	any	any	only Linux based	only Linux based

Table 3.1: Comparison of virtualization and containerization solutions

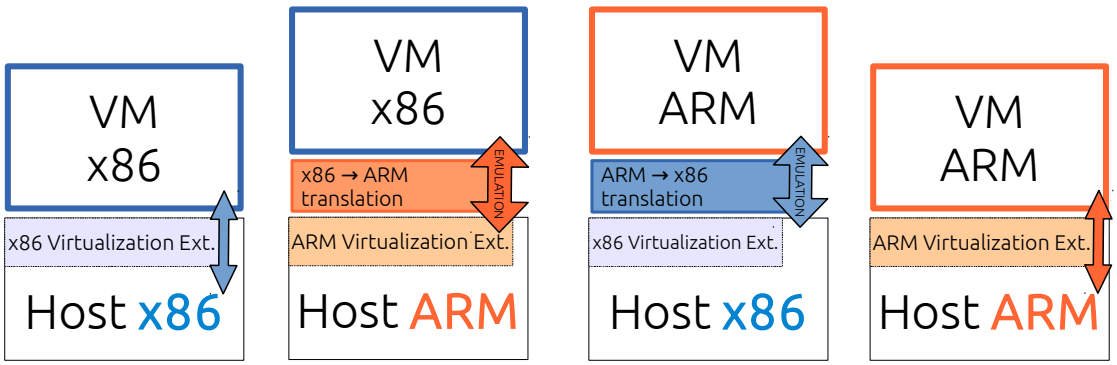
This comparison leads us to further study the virtual machine solution. It is the most common approach in data centers and also the most generic solution as it imposes the least number of restrictions. KVM was our favorite technology for its easy availability in the Linux kernel and because it is closely linked with *QEMU*, short version of *Quick Emulator*. This technology can run a variety of unmodified guests by emulating the guest hardware and processor with *binary translation*. The way *QEMU/KVM* works is the following: if the host and the guest architectures are the same and the host features virtualization extensions, KVM uses them and run the guest at a near-native speed; if the guest has a different architecture from the host, QEMU uses binary translation to do emulation, which implies some computation overheads. We focus on this emulation technology as it appears to be a solution to make ARM and x86 architectures cohabit in the same data center. But first we need to measure and evaluate the emulation overheads to see if they are bearable costs or not.

3.3 Performance Comparison between ARM and x86

3.3.1 Intended Use of Virtualization and Emulation

We have seen that virtualization extensions are available on both ARM and x86 architectures. But as we propose to gather both types of machines in the same infrastructure, we want to be able to migrate virtual machines across all physical machines. The architecture of the virtual machine is fixed and can not change during its execution. The idea is that this virtual machine is executed using virtualization extensions if it running on the right architecture, while executed via emulation when it is on a different architecture. Indeed, emulation allows to translate a binary code written in one instruction set, the source, towards another instruction set, the target. As QEMU/KVM automatically detects if the hardware has virtualization capabilities, our assumption is that when migrating a virtual machine between two different types of host, the software will automatically switch from emulation to virtualization extensions, or the opposite.

Figure 3.1 pictures the two alternatives for the virtual machine architecture and their underlying functioning. First and last cases have low or no overhead thanks to the virtualization extensions, while the two cases needing binary translation suffer from performance degradations due to emulation.



(a) x86 VM (b) ARM VM

Figure 3.1: Two alternatives for virtual machines architectures: using emulation or virtualization extensions

We lead a performance study of emulation for both architectures in order to decide which solution would be the best. As emulation is very compute intensive, we already supposed before starting the experiments that emulation on ARM host would not provide excellent results. Giving the high performance of x86 servers, our hope was that emulating the ARM instruction set would not be too costly and would still provide better results than a native ARM processor. Nevertheless we wanted to measure the performances of both alternatives in order to have a complete study.

The emulation software QEMU features different operating modes. The first one is *system emulation*, which emulates a full computer system, including its peripherals, running any guest OS. The second one is *user-mode emulation*, which only runs programs

that were compiled for a different instruction set. Even if system emulation of x86 processor is available on the ARM version of QEMU, we have never managed to successfully boot an x86 based virtual machine on the ARM processor we used. Consequently, to be able to complete a full study of both sides, we decided to use user-mode emulation and compare two types of executions: native or emulated, of two program versions: compiled for ARM or x86.

3.3.2 Selected Hardware and Setup

ARM Cortex-A15 is the chosen processor to represent ARM architecture as it is the first to implement virtualization extensions. It was first available in the *Samsung Chromebook* in 2012, and inside the *HP 11 Chromebook* in 2013, which both feature the same system-on-chip (SoC), the *Samsung Exynos 5250*, containing a dual core ARM Cortex-A15 with a frequency of 1.7 to 2 GHz. Note that Cortex-A15 is the same processor as the *"big"* processor of *ARM big.LITTLE*. As their code names suggest, these notebooks come with *Google Chrome OS*. To be able to use KVM software and benefit from the virtualization extensions it is necessary to install a Linux distribution, and in particular the Linux kernel version must be equal or posterior to 3.9. In fact, Chrome OS is already a Linux-based OS, but for these Chromebook it uses a kernel version of 3.8.11, not supporting virtualization extensions. Moreover, installing Linux on ARM platforms is not as easy as it can be for x86 architecture. ARM systems lack ways for hardware discovery such as a standard *BIOS* or *PCI* bus. As a consequence, Linux kernel developers must write specific files, called *device tree sources (dts)* describing each ARM hardware.

At the time we worked with the HP Chromebook, it was not supported by Linux as its dts file did not exist yet [1]. Instead we made it run an Ubuntu distribution based on the same kernel of the Chrome OS already installed. To measure its power consumption, we use the *powerstat* Ubuntu package which retrieves monitoring data from the battery via the *Advanced Configuration and Power Interface (ACPI)*.

On the contrary, Samsung Chromebook was already supported because as the first available device with the ARM Cortex-A15 processor, it was of great interest for developers to be able to use virtualization extensions. We have installed on the Samsung Chromebook an Ubuntu version 12.04 with a Linux kernel 3.13, and installed QEMU 2.0 with KVM acceleration capabilities. We used an external watt-meter called *Plogg* to measure the instantaneous power consumption of the machine.

We have selected x86 servers available on the Grid'5000 testbed for our experiments. Grid'5000 is a French experimental platform, geographically distributed over 11 sites in France and Luxembourg, dedicated to scientific research concerning large scale infrastructures [10]. It is very convenient as some clusters are equipped with power monitoring systems accessible via an API named *Kwapi* [49]. We have chosen an *Intel Xeon* processor and an *AMD Opteron* from monitored clusters located respectively in the cluster named *Taurus* of Lyon and the *Parapluie* cluster of Rennes. We find relevant to select two kinds of x86 servers because it allows to highlight the possible differences between two generations and two constructors of quite similar servers. Both servers run a Debian Wheezy

[1] HP Chromebook 11 is now supported by Linux as its dts file was included in mainline Linux kernel in 2015.

operating system with QEMU 1.7 installed. In Lyon, electrical consumption is acquired with wattmeters from *Omegawatt*, whereas in Rennes monitored *Power Distribution Units (PDU)* from *EATON* are used and power data is fetched via *SNMP* requests.

Characteristics of the hardware are gathered in Table 3.2. One striking point is the huge difference between idle consumptions. *Parapluie*'s idle power is more than 20 times greater than the *Chromebook*'s, and 2 times greater than *Taurus*'s. The upper bound corresponds to the maximum power consumption measured when the processor is fully loaded, using as many instances of *cpuburn* benchmark as the number of cores.

Codename	Chromebook	Taurus	Parapluie
Fullname	Samsung / HP 11 Chromebook	Dell PowerEdge R720	HP Proliant DL165 G7
Architecture	ARMv7 32 bits	x86 64 bits	x86 64 bits
CPU	1 x Cortex-A15	2 x Intel Xeon E5-2630	2 x AMD Opteron 6164
Total Nb of Cores	2	12	24
Range of Power Consumption	5 – 25 W	96 – 227 W	180 – 280 W
Release year	2012 / 2013	2012	2010

Table 3.2: Summary of selected hardware and their characteristics

3.3.3 Benchmark Results: Native vs. Emulated Performances

Performance Only

We made experiments with *QEMU User Emulation*, a tool which allows to execute binaries compiled for a different architecture by dynamically translating the instructions during the execution. Not all types of program can be executed through dynamic translation, it needs to be an application compiled with statically linked libraries. We have selected the synthetic benchmark program *nbench* [40]. It is a simple program written in C, which is composed of several subprograms designed to test CPU capabilities of a machine. Among those subprograms, we have chosen the *IDEA encryption* benchmark to evaluate integer computation, and the *Fourier coefficients* algorithm for float computation. We have compiled the program with static libraries in two versions: one compiled for ARM architecture, the other for x86. We have executed these two versions on each selected machines: natively when the architectures of the host and the program are the same, and with QEMU in user emulation mode for binary translation when the architectures differ.

Table 3.3 shows the native and emulated performances for the three machines previously described. Results are expressed as the maximum number of iterations per second reached during the benchmark execution. The column "OVERHEAD" represents the ratio between emulated and native performances.

	Native (Iterations/sec)		Emulated (Iterations/sec)		Overhead Native/Emulated	
	Int	Float	Int	Float	Int	Float
Chromebook (ARM)	8233,9	27 251	932,46	604,07	8,83	45,11
Taurus (x86)	102 893,9	380 437	11 479,22	11 153,06	8,96	34,11
Parapluie (x86)	113 569,8	320 823	15 239,46	12 599,76	7,45	25,46

Table 3.3: Native vs Emulated performances for each hardware. Colum "Int" refers to *IDEA encryption*, and "Float" to *Fourier coefficients* of nbench benchmark, while "Overhead" represents the ratio between emulated and native performances.

We realize that the order of magnitude of the overhead is the same no matter the underlying physical architecture. For integer computation the emulation is between 7 to 9 times slower, while for float computation the overhead is much more important, from 25 to 45 times, the largest being not surprisingly for the ARM processor of the *Chromebook*.

Even if x86 processors are natively more powerful than ARM processors, (about 12 to 13 times in our selection) the important overhead causes the emulation to slow down a lot all the processors. We can even notice for float computing that the ARM native execution is in fact more powerful than the emulated execution on x86 servers. Indeed, if we consider the ARM compiled float benchmark, the *Chromebook* reaches 27 251 iterations per second whereas *Taurus* and *Parapluie* reach respectively 11 153,03 and 12 599,76 iterations per second.

Performance and Power Consumption

As we are aiming at reaching energy proportionality, we are not only interested in performances but also in the associated power consumption. That is why we measured the power consumed by the machines during benchmark executions. Figures 3.2 and 3.3 represent the average instantaneous power consumption for hardware over an evolving performance level from 0 to maximum, expressed in number of iterations per second of the IDEA encryption benchmark. The starting point of each curve corresponds to the average power consumption in idle state, and the ending point to the average power consumed during a complete execution of the benchmark. To obtain intermediate data points, we have slightly modified the nbench benchmark by introducing *nanosleep* calls in order to reduce the maximum performance. The benchmark is run five times with five different durations of sleep, resulting in five data points for each hardware, approximated with a linear fitting to get the final curves.

Each graph plots three curves corresponding to our three selected hardware presented in Table 3.2. The most powerful is the *Parapluie* server, and it defines the maximum scales of our graphic. The two other curves are also endless because we repeat the power consumption profile to simulate the fact of having multiple servers of each type. The least powerful hardware is the ARM *Chromebook*, but because of its very low consumption it can be repeated several times and still fit in the graph. The maximum performance of one single *Chromebook* is symbolized by the vertical dashed line. On the opposite, when we repeat server *Taurus*, it shortly becomes out of scale because its static idle consumption is too important.

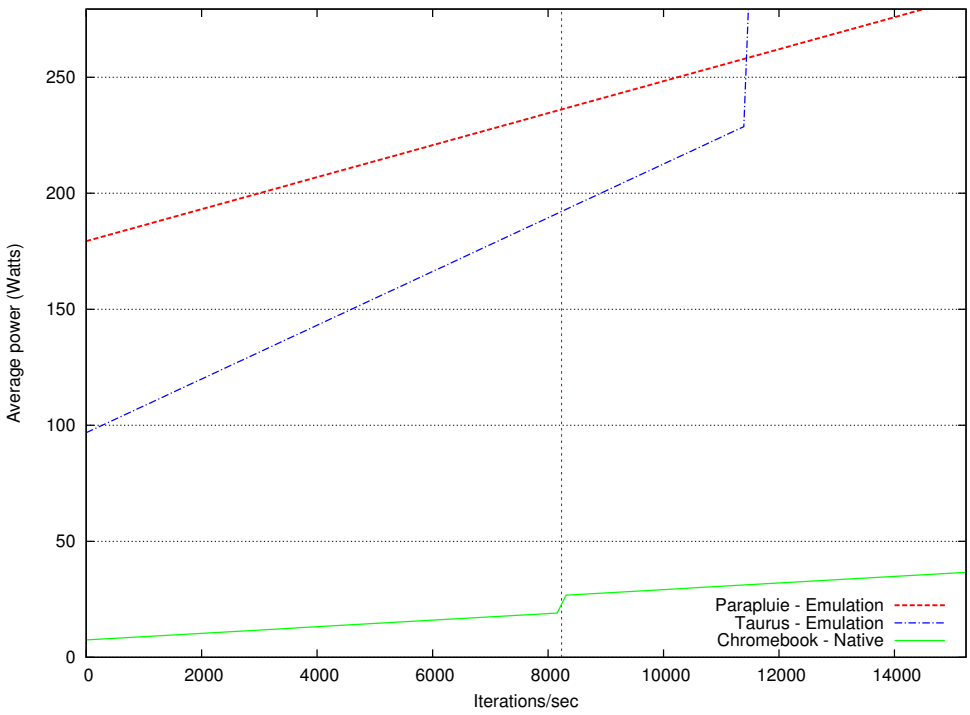


Figure 3.2: Average power consumption (Watts) regarding the number of iterations per second of the same ARM program (IDEA benchmark) on 3 different hardware devices

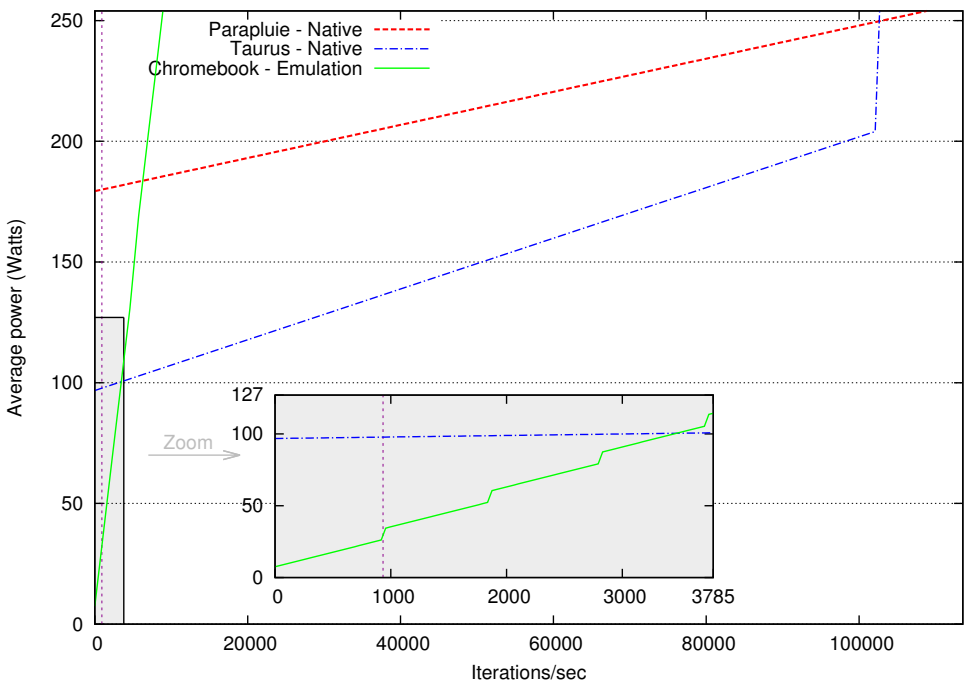


Figure 3.3: Average power consumption (Watts) regarding the number of iterations per second of the same x86 program (IDEA benchmark) on 3 different hardware devices

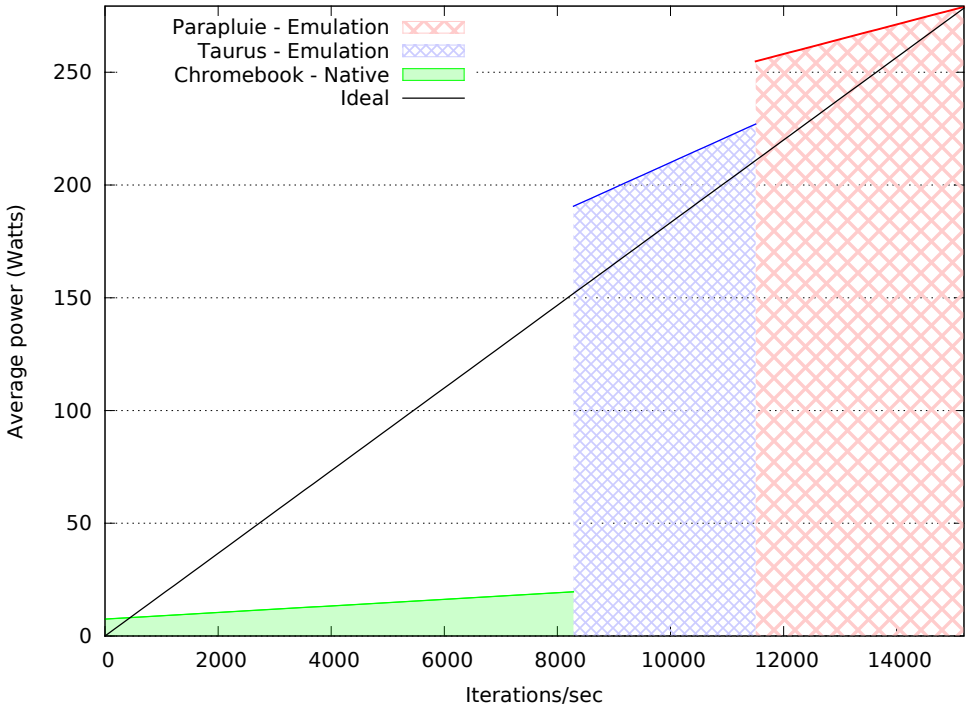


Figure 3.4: ARM solution compared to ideal proportionality (IDEA benchmark)

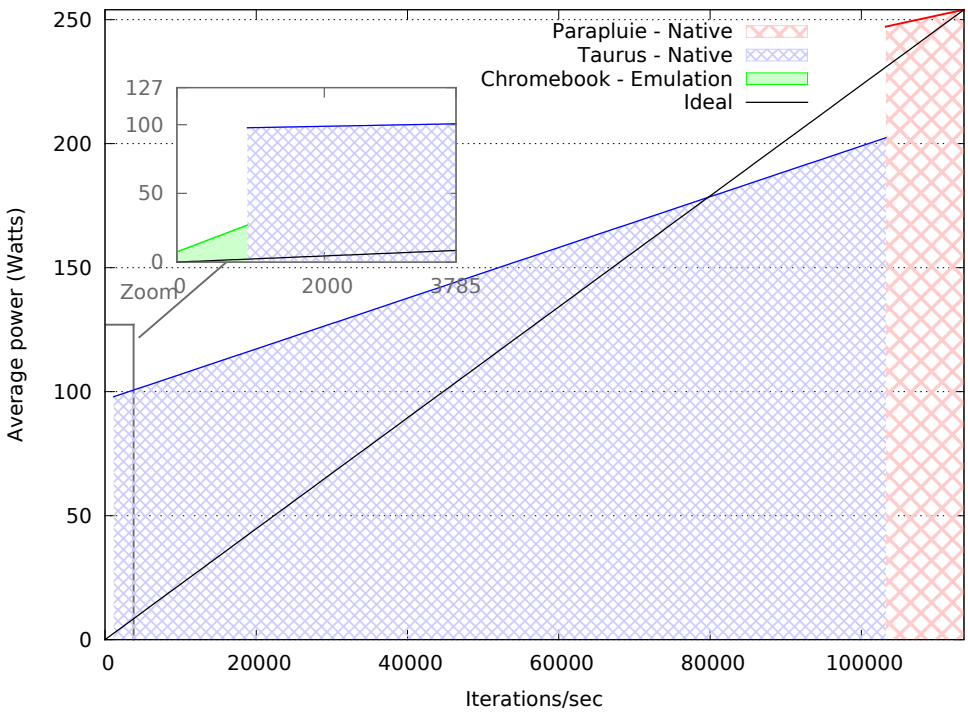


Figure 3.5: x86 solution compared to ideal proportionality (IDEA benchmark)

Figure 3.2 corresponds to the case depicted in Figure 3.1(b) where the executed program is compiled for ARM architecture. The program is executed natively on *Chromebook* and through dynamic translation on *Taurus* and *Parapluie*. On the opposite, Figure 3.3 represents what happens in the case of Figure 3.1(a) where the target architecture is x86 and the emulation only concerns *Chromebook*. When we compare the two graphs, and especially when we observe the maximum number of iterations per second, we can find the overhead of emulation discussed in Table 3.3. The overall total performance is reduced by 7.45 times when we use an ARM binary.

For the ARM alternative on Figure 3.2, we see that x86 architectures offer poor performance for power ratios because if the execution of the program could be distributed on two ARM machines, then it would always be the most relevant configuration concerning energy consumption. If we cannot consider program parallelization, then *Taurus* would be the chosen platform from approximately 8000 to 110 000 iterations per second, and *Parapluie* would be the most efficient passed this level of performance. This creates a result not far from energy proportionality as we observe on Figure 3.4.

On the other hand, for x86 program on Figure 3.3, the performance of ARM platform is very low as it is drastically reduced due to dynamic translation. Consequently, what we can observe on the zoom area is that the *Chromebook* would be chosen until about 900 iterations per second if no parallelization, and until approximately 3600 iterations per second, which represents 4 *Chromebook* nodes, if parallel execution is possible. Considering the last perspective, the energy consumption is thus reduced for the first 1/30th of the total performance, and the global shape is still far from proportionality as shows Figure 3.5.

With Figures 3.4 and 3.5, we want to picture how far from proportionality this solution could be. For each level of performance, it shows only the most energy efficient hardware. We also plot an ideal curve which starts from 0 and is strictly proportional until the maximum point, corresponding to the average power consumption when the most powerful hardware of our platform reaches its maximum number of iterations rate. Only a 1-1 relation is considered here as if parallelization would not be possible.

When analyzing the ARM alternative pictured in Figure 3.4, we can see that the ARM hardware leads to important energy savings. In fact, its curve is way under the ideal, except for the very beginning because its idle power consumption is not equal to zero. Moreover, having these two different x86 servers is also a good leverage and allows to better stick to the ideal proportionality line. This confirms the assumption we made when selecting two different kinds of x86 hardware, and we can interpolate and imagine that even more recent servers would add more proportionality. Nevertheless, even if this alternative seems not far from proportionality, it is not optimal because it wastes the potential performance of x86 servers.

In the x86 alternative shown in Figure 3.5, the gains from ARM hardware are only profitable for a reduced part of low performance, that we can only see on the zoomed part of the graph. The most predominant hardware is *Taurus*, and we realize that *Parapluie* only brings a small improvement in performance but consumes a lot more than *Taurus* most of the time. This can be justified by the fact that the Dell PowerEdge R720 (i.e., the Taurus hardware) is the most recent server of our selection, and the energy efficiency aspect must have been better considered during its design.

3.4 Virtual Machine Migration between ARM and x86

We have performed some experiments of live migration with an ARM based virtual machine. As previously explained, we could only use an ARM virtual machine because x86 emulation on ARM devices with QEMU was not fully effective at the moment. We have built an ARM virtual machine with Debian Wheezy OS that emulates the board *Versatile Express A15* containing one virtual ARM Cortex-A15 processor, with a 4 Go virtual disk. As it has one virtual CPU, it only executes itself on one core of each host machine. We used *Libvirt* version 1.2.9 as VM manager. Hardware used is an *HP 7800* server with an *Intel Xeon E5620* CPU, and the ARM *Samsung Chromebook*. They are both monitored with external wattmeters *WattsUp Pro* and power data is acquired and stored with Kwapi API [49].

Due to technical limitations, we only managed to migrate the virtual machine from the x86 server to the Chromebook, and not the opposite way. The virtual machine is first started on the server where it is executed thanks to emulation. The migration is launched with *Libvirt*, and as soon as the virtual machine is fully migrated on the *Chromebook*, it is executed using KVM and the virtualization extensions. The virtual machine was running a web server serving static contents, which stays operational after the migration.

Figure 3.6 presents the dynamic power consumption of each host during the process of virtual machine migration. To be able to compare the two curves more easily, we substract the idle consumption to the measured consumption of each machine to show only the impacts on dynamic power (The idle consumptions are respectively 5 Watts for Chromebook and 149 Watts for HP 7800 server).

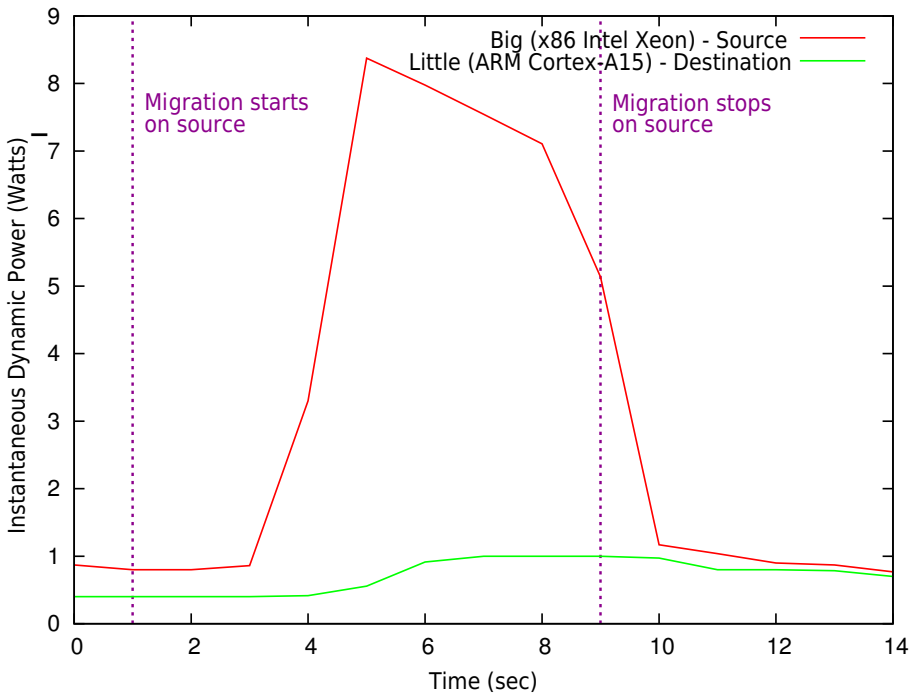


Figure 3.6: Dynamic power consumption (Watts) during live migration of ARM virtual machine from Big (HP 7800 Workstation) to Little (Samsung Chromebook)

The live migration duration is 8 seconds for this example, which corresponds to a data transfer of 53 Megabytes. The two physical machines are linked with a 1GB switch and cables, but as the *Chromebook* does not have an Ethernet port, we use an Ethernet to USB 2.0 adapter which may reduce the network throughput. Concerning power consumption, we notice a significant overhead for the source host, about 9 watts when starting the migration. On the destination host there is an increase in power consumption when receiving the virtual machine but then the power stabilizes shortly.

3.5 Discussions and Conclusions

The benchmarks results shown in this chapter confirm the interest of having both ARM and x86 architectures in the same infrastructure to approach energy proportionality. Indeed, the ARM processor we have studied, *ARM Cortex-A15*, has a very low power consumption compared to standard x86 servers. But not surprisingly its performances are also lower, especially for float computation. It results in a large performance gap between ARM Cortex-A15 and x86 servers for this type of applications. This gap can be reduced if parallel execution on multiple ARM nodes is possible. Additionally, it could be interesting to broaden the range of architecture types and look for ones that fit inside the gap regarding performance and energy consumption. Our intuition is that *ARMv8 64bits* architecture, which promises better performances than Cortex-A15 (*ARMv7 32 bits*) could allow to get even closer to energy proportionality.

We have explored possible solutions to use the two architectures in a transparent manner. We consider two main categories: virtual machines and containers. We focus on open source solutions, that is why we selected KVM and Xen hypervisors for virtualization, and LXC and OpenVZ for containers. Even if containers have recently gained much attention from the cloud community, they do not seem appropriate to fulfill our goal. We put our focus on virtualization and emulation to benefit from the recent virtualization extensions introduced by ARM. Unfortunately, the emulation performances are disappointing, and especially because we did not expect the emulation overhead to be this important for powerful x86 servers. Maybe the technology can be improved in both translation ways. A Russian company, *Eltechs*, is commercializing a product to run x86 applications on ARM devices, and claims that it is 4.5 faster than QEMU, but we could not test it as it is a proprietary solution.

Our first experiments on virtual machine migration between ARM and x86 were quite promising, but they remained at a proof of concept stage. All these experiments were done in early 2014, and we were working with recent technologies, that were still in development and need to be enhanced, which requires a lot of engineering work.

For the moment, we found that the best solution is to run applications natively on all hardware to benefit from their full performances to be able to reach energy proportionality. Consequently, we decided to work with applications that can be compiled and executed on both types of architectures, have the ability to be migrated with no or few constraints, and support parallel distribution. That is why in the following work presented in this thesis, we focus on stateless web servers. We detail the characteristics of the chosen type of application as well as the specifications of our proposed framework in the next chapter.

– Chapter 4 –

Design of "BML": Energy Proportional Data Center

This chapter details our framework with all its specifications. We particularly focus on the infrastructure part by describing its building process step by step. Then we give an implementation of the infrastructure with existing heterogeneous hardware and provide its energy proportionality evaluation at the server level.

4.1 General Overview of BML Framework

The framework we propose is called "**BML**", for "**Big, Medium, Little**". It is meant to handle the management of a heterogeneous cluster, from application placement to resource reconfigurations with the objective of energy proportionality. This name highlights the inspiration from *ARM big.LITTLE*, but with the introduction of the term *Medium*, we generalize it to any number of architectures. Indeed, *BML* does not signify that only three types of machines are considered, but that we take into account a set composed of multiple architectures that each has a different range of performance and energy consumption characteristics.

Figure 4.1 shows the connections between the different components of the BML framework. On the left, the different architectures are gathered in a heterogeneous infrastructure. We explain in details in section 4.3 how each type of hardware needs to be profiled, and how these profiles are used to compute the ideal energy proportional combinations of machines. On the right of Figure 4.1, the application and its load profile are depicted. The evolution of the load over time may be known a-priori or predicted as we discuss in section 4.2, together with other specifications regarding applications characteristics.

The general functioning is the following: the load prediction module gives as output a prediction concerning the performance rate that the application will require at a certain point in the future. Given this information, the scheduler consults the BML combination module to compute the ideal combination of machines needed to achieve this performance rate. Then, depending on its settings, the scheduler can decide to perform some reconfigurations towards this new combination. If such a decision is taken, the reconfiguration plan is transferred to the resource manager, which is responsible for triggering all recon-

figuration actions. These actions consist in switching on and off the machines involved and migrating the applications from one or more sources to one or more destinations.

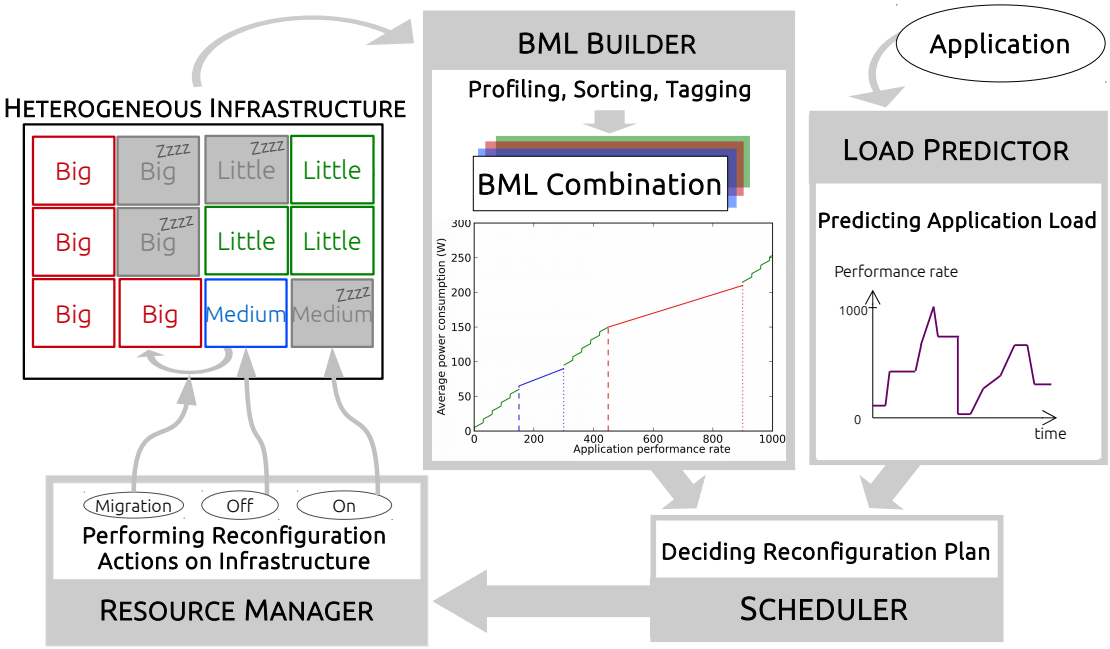


Figure 4.1: BML Framework Overview

Of course, this framework is a generic overview, and each module implementation can vary in many possible ways: the architectures characteristics, the applications, the load prediction system, but also the scheduling and resource management policies. We give an implementation of the infrastructure in section 4.4, and two scheduler implementations with their evaluations in the chapters 5 and 6.

4.2 Characterizing the Application and its Load

We target applications with *variable load* over time and we aim at adapting the computational capacity of the infrastructure to load conditions so that the energy consumption more closely matches resource utilization. Load variations are particularly present in cloud web services, but also often appear in HPC applications [11]. For such goal, the application performance needs to be characterized using an *application metric*. This metric represents the amount of work produced by the application over a given time unit, that is why we sometimes refer to it as *performance rate*. It is important that this metric is unique and independent of the underlying architecture running the application. BML framework seeks to minimize the energy consumption by dynamically taking provisioning decisions, but the energy consumption is not to be reduced at the expense of performance.

The *Quality of Service (QoS)* required by the application is carefully taken into account, and the intended QoS level directly impacts the relevance of reconfiguration decisions. Applications can be classified regarding their performance and QoS requirements. They can either be *critical*, if applications have stringent performance requirements, or *tolerant*, if they have soft QoS requirements. For example, *critical* applications can be found in banking and medical areas where delays have serious consequences. More *toler-*

ant applications are found in enterprise services, or services with flexible deadlines such as backup systems. It is possible that certain applications lie in between these two classes, and hence, depending on the use-case, several intermediate classes be defined.

An important characteristic is the application *malleability*, i.e., as defined in [42] its ability to be executed in a distributed manner across several processors or machines. In some cases, the malleability can be constrained, and the minimum and maximum number of application instances should be specified. This criterion poses a constraint when computing the possible machine combinations for running the application. If the application is fully malleable, meaning that the number of instances is not limited, the infrastructure can be exploited without constraints. It is also necessary to determine the ability for the application to be migrated across machines. This is defined by how the application maintains state and on the amount of data to transfer. In case of stateful applications, the migration overhead should be evaluated, both in terms of duration and energy consumption.

The knowledge of the load evolution is a crucial parameter in our system as it has a direct impact on the relevance of the resource reconfigurations. This knowledge can be *perfect* if the load can be determined with a sure precision. It can be *partial* if certain characteristics are known, such as weekly, diurnal, hourly patterns, but the accuracy of load variations is not precise. Finally the load can be *unknown* if no a priori information is available. In such case, the load must be predicted for future intervals using for example learning techniques upon historical data.

4.3 Building BML Infrastructure Step by Step

Through this section we detail step by step the process we use to build a BML infrastructure, which corresponds to the *BML builder* module of Figure 4.1. It begins with the profiling of hardware and ends with the computation of ideal combinations of machines. We consider a scenario where multiple machine types are available to choose from, and there is no limitation in the number of machines of each type. This scenario enables to create perfect combinations of machines, and is equivalent to a capacity planning where a data center infrastructure is built specifically for the purpose of the target application with the objective of being energy proportional. With some minor changes, this work can be adapted to consider cases where a heterogeneous infrastructure has already been established, and there are thus limited numbers of machines of each type.

The first step towards building an energy proportional infrastructure is to determine the energy consumption and performance characteristics of the available hardware. Hence, each machine type needs to be profiled while running the target application to evaluate the maximum performance rate it can reach and the amount of energy it consumes. The overheads of switching on and off a machine are also part of its characterization profile. Both the time and the energy consumption required during these actions are measured. Once all profiles are built, a computational phase is conducted to build the ideal machine combinations to achieve energy proportionality.

To ease the step by step explanations, we illustrate the process with four theoretical examples of architectures as input. However, this methodology is generic and can work with n different types of architecture.

4.3.1 Step 1: Characterizing Each Architecture Profile

A profile characterizes the behavior of an architecture in terms of power consumption and performance when running the target application. A machine profile contains at least two data points, namely its idle power consumption and its energy consumption at maximum performance. This maximum performance rate is expressed with an *application metric* that represents the amount of work performed over a given time step denoted Δt. In our experiments, we use as metric the number of requests processed per second by a web server, but for example it can be frame rate for an application of video rendering.

To achieve an energy proportional infrastructure, the machines are dynamically powered on and off to reduce the impact of their static power consumption when their are inactive. This implies to measure the time duration and energy consumption of the switch on and off actions for each machine type, in order to take them into account when making reconfiguration decisions.

Following is a summary of the data we acquire during the profiling phase. Let $\mathcal{M}$ be the set of architectures available to compose our heterogeneous infrastructure, for each architecture $i \in \mathcal{M}$, we collect from profiling:

- ${perf_{max}}^i$: maximum performance rate reached by architecture i, expressed regarding an *application metric*.
- ${power_{max}}^i$: average instantaneous power consumed by architecture i when achieving ${perf_{max}}^i$, expressed in Watts. If the time step Δt is not equal to one second, ${power_{max}}^i = {e_{max}}^i/\Delta t$, (where ${e_{max}}^i$ is the energy consumed by architecture i during one time step Δt when achieving ${perf_{max}}^i$, expressed in Joules).
- ${power_{idle}}^i$: average instantaneous power consumed by architecture i in idle state, expressed in Watts.
- ${t_{On}}^i$: time required to power on architecture i, expressed in seconds.
- ${t_{Off}}^i$: time required to power off architecture i, expressed in seconds.
- ${e_{On}}^i$: energy consumed during the power-on period of architecture i, expressed in Joules.
- ${e_{Off}}^i$: energy consumed during the shut-down period of architecture i, expressed in Joules.

Given this information and assuming that the power consumption is linear between ${power_{idle}}^i$ and ${perf_{max}}^i$, a function, named $powerFor^i$, is created to compute the power consumed by a given architecture i for the specified performance rate $perfRate$ given in input. The assumption we make on linear power consumption might lead to small under- or over-estimation, as studied by Rivoire *et al.* [21]. Yet, this approximation is precise enough for our solution, and eases the profiling phase. Although acquiring more intermediate data points, if the application allows, or considering potential DVFS, would enable more precision, our methodology would not be affected.

Algorithm 1 details the function $powerFor^i$, where the computed power represents the instantaneous power consumption of the needed number of machines of architecture i

to provide the performance rate $perfRate$. If another model than linearity is considered for energy modeling, then this function would have an increased complexity.

Algorithm 1 $powerFor^i$: Computes instantaneous power consumption for a given architecture i and a given $perfRate$

Input: Architecture: i, Performance rate: $perfRate$
Output: Power consumption: $totalPower$

1: $nbFullNodes \leftarrow \lfloor\ perfRate\ /\ perf_{max}{}^i\ \rfloor$
2: $remainingRate \leftarrow perfRate - (\ nbFullNodes \times perf_{max}{}^i\ )$
3: $remainingPower \leftarrow 0$
4: **if** $remainingRate > 0$ **then**
5: $\quad slope \leftarrow (\ power_{max}{}^i - power_{idle}{}^i\ )\ /\ perf_{max}{}^i$
6: $\quad remainingPower \leftarrow (\ slope \times remainingRate\ ) + power_{idle}{}^i$
7: **end if**
8: $totalPower \leftarrow (\ nbFullNodes \times power_{max}{}^i\ ) + remainingPower$
9: **return** $totalPower$

Figure 4.2 gathers four different profiles of theoretical architectures named A, B, C, and D that we use to illustrate the explanations. On each graph is plotted the result of the function $powerFor^i$. All four graphs have the same scale. On each of them a vertical line represents the maximum performance of the machine type, and beyond this line is the power consumption of multiple machines of the same architecture type. We consider for the following steps that the set of available architectures is $\mathcal{M} = \{A,B,C,D\}$.

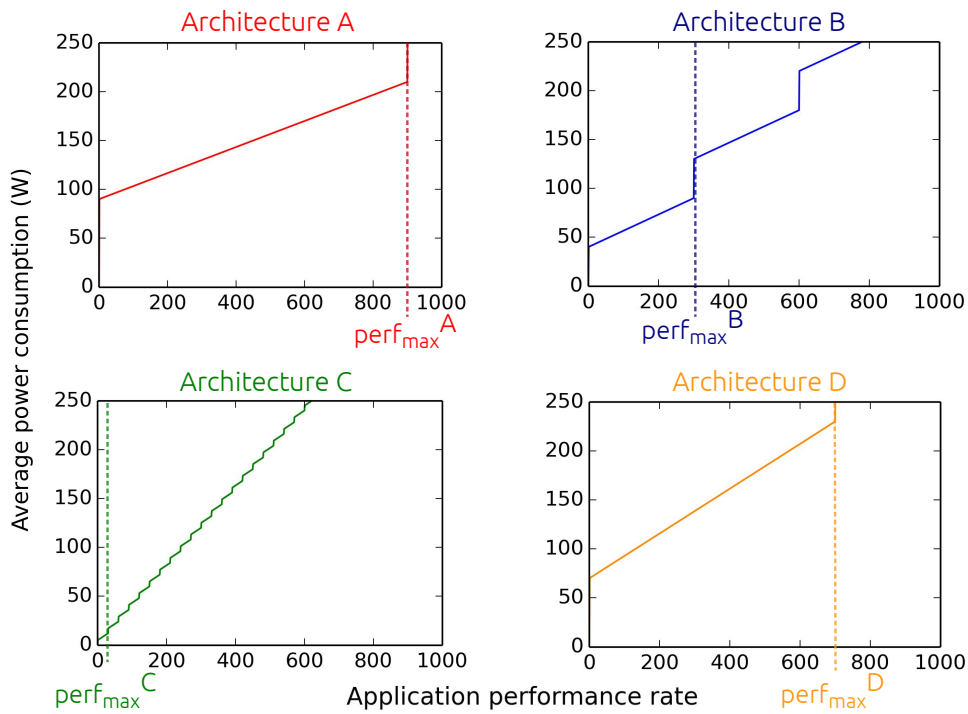


Figure 4.2: Power and performance profiles of 4 illustrative architectures A, B, C and D. Vertical line is the maximum performance of one machine, and beyond is the cumulated power for multiple machines of the same architecture.

Concerning switch on and off actions, it is fundamental to consider their overheads and understand in which conditions it is beneficial to perform such an action and when it is not. Indeed, we aim at saving energy by turning off unused machines, but we also want to respect QoS requirements, that is why we need to power on machines perfectly in time to process the incoming workload. As a consequence, the decision of turning off a machine must be taken carefully, with knowledge that this machine will stay powered off for a certain minimum interval.

For this purpose, we define the **Minimum Switching Interval**, noted T_s^i, as the minimum amount of time for which it is more efficient to switch off machine i than to keep it powered on but idle. It is computed as follows:

$$T_s^i = \max\ \left(\frac{{e_{On}}^i\ +\ {e_{Off}}^i}{{power_{idle}}^i}, {t_{On}}^i + {t_{Off}}^i \right)$$

This interval is inspired from the definition of Orgerie *et al.* in [44], while assuming that P_{OFF}, the consumption of a machine when it is powered off, is equal to 0. We need to enhance the definition by adding the maximum function because with some low power processors we have profiled, it happens that $t_{On} + t_{Off}$ is actually greater than the fraction in the first term of our T_s definition. In [37], Lu *et al.* describe a similar concept, but in a more theoretical way, as Δ which they called *critical time*. We explain how we use T_s in the scheduling algorithm described in Chapter 6.

4.3.2 Step 2: Sort Architectures to Keep Only BML Candidates

After having profiled all the architectures, the next step consists in analyzing these profiles and verifying that all types of machines are appropriate to get close to energy proportionality. This starts by sorting machines according to their maximum performance in decreasing order. Then we verify if power consumption respects this initial ordering. We proceed by comparing sorted architectures in pairs; if an architecture has lower performance than another while consuming more energy, we remove it from the BML candidates as it does not possess the required properties to improve energy proportionality. This process is detailed in Algorithm 2, which is generic and can work with n different types of architecture. A list with the relevant architectures for building a BML infrastructure is available at the end of this step.

Figure 4.3 gathers the profiles of the four architectures A, B, C, and D. After executing Algorithm 2 on this set, only three architectures are kept as good candidates for a BML infrastructure, namely A, B and C. Architecture D is discarded because its maximum power consumption is greater than that of A, which is the most powerful machine, i.e., ${perf_{max}}^D < {perf_{max}}^A$ but ${power_{max}}^D > {power_{max}}^A$. This implies that architecture D would not help increase the energy proportionality of our infrastructure. We can keep A, B and C because they respect the same ordering for performance and power consumption: ${perf_{max}}^A > {perf_{max}}^B > {perf_{max}}^C$ and ${power_{max}}^A > {power_{max}}^B > {power_{max}}^C$.

Once this initial filtering is complete, architectures are sorted according to their performance so that each of them can be labelled as *Big*, *Medium* or *Little* to increase readability. For our illustration, the result is: A $\leftarrow$ *Big*, B $\leftarrow$ *Medium*, and C $\leftarrow$ *Little*.

Algorithm 2 *candidatesBML*: Sorts architectures and keeps only good candidates for BML infrastructure

Input: Architectures: $archList$
Output: BML candidates: BML

1: $BML \leftarrow [\,]$
2: $archList.sort(perf_{max}{}^{i}, descending)$
3: $previous \leftarrow archList[0]$
4: $BML.append(previous)$
5: **for** $i \in [1, archList.length - 1]$ **do**
6: $\quad current \leftarrow archList[i]$
7: $\quad$ **if** $power_{max}{}^{current} < power_{max}{}^{previous}$ **or** $power_{idle}{}^{current} < power_{idle}{}^{previous}$ **then**
8: $\quad\quad BML.append(current)$
9: $\quad\quad previous \leftarrow current$
10: $\quad$ **end if**
11: **end for**

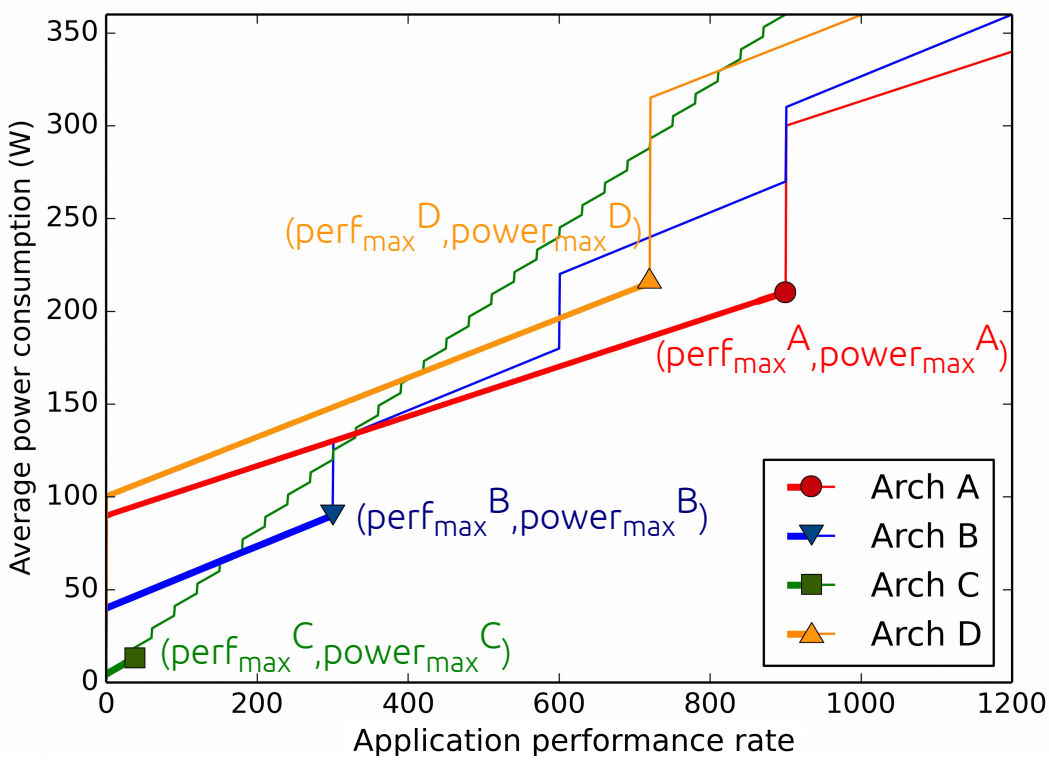


Figure 4.3: Architectures A, B and C are good candidates for BML infrastructure, but Architecture D will be removed from consideration due to its poor energy efficiency compared to architectures A, B and C.

4.3.3 Step 3: Finding Crossing Points between Architectures

This step determines how chosen architectures should be combined to create the most power proportional infrastructure. We define a *minimum utilization threshold*, denoted $minThreshold^i$ for each architecture $i \in \mathcal{M}$, expressed regarding the application performance metric. For instance, if there are two architectures, i as *Little* and j as *Big*, then the minimum threshold of architecture j corresponds to the point from which its power consumption becomes more relevant than i's for the considered performance rate. Initially, all minimum thresholds are set to 1. Whilst this threshold will remain 1 for the *Little* architecture, as it has the lowest idle power consumption, the function described in Algorithm 3 will compute the thresholds for all remaining architectures. These points where an architecture becomes preferable over another are also termed as *crossing points* as they represent the points where power profiles meet.

Figure 4.4 illustrates this step with architectures A, B and C, now denoted *Big*, *Medium* and *Little*. The utilisation threshold of *Medium* starts around a performance rate of 150. Before this point, it is more efficient to use up to five *Little* nodes. The minimum utilization threshold of *Big* architecture corresponds to the maximum performance rate of a *Medium* node. A substantial jump in power consumption results from switching from *Medium* to *Big* since this crossing point is not optimal. The next step is needed to improve it.

Algorithm 3 $CrossPoints_{Step1}$: Finds crossing points between architectures

Input: BML candidates: BML
Output: Crossing points: $crossPoints$, Updated BML candidates: BML

```
1:  LMB ← BML.reverse()
2:  crossPoints ← [ ]
3:  j ← 1
4:  for i ∈ [0, LMB.length − 2] do
5:      current ← LMB[i]
6:      next ← LMB[i + 1]
7:      while j ⩽ perf_max^next
8:          and powerFor^current(j) < powerFor^next(j) do
9:          j ← j + 1
10:     end while
11:     crossPoints.append(j)
12:     minThreshold^next ← j
13: end for
```

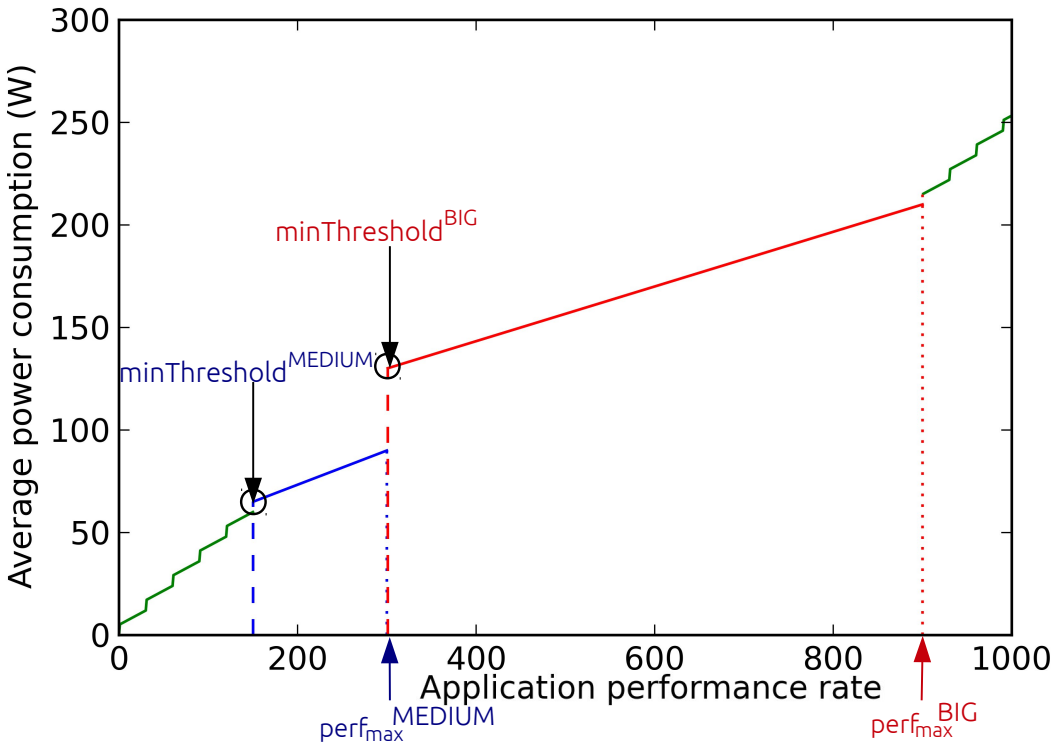


Figure 4.4: First step of crossing points computation between *Little* and *Medium*, and between *Medium* and *Big*

4.3.4 Step 4: Finding Crossing Points between Architectures and Combinations of Smaller Architectures

This step is required when there are more than two architectures. The previous step computes the crossing points between homogeneous combinations of machines, but with three architectures and more, one must determine whether adding *Little* nodes to *Medium* combinations help improving power proportionality and reduce the gap between *Medium* and *Big* architectures. The function, detailed in Algorithm 4, re-evaluates the minimum utilization thresholds of the most powerful architectures of the infrastructure. For our illustrative example, it re-computes the crossing point between *Medium* and *Big*. Of course the minimum threshold of *Little* stays at 1, and the one of *Medium* can not change because only homogeneous combinations of *Little* nodes can be used before switching to a *Medium* node. Algorithm 4 calls the function $idealBML$ detailed in next step in Algorithm 5, which computes the combination of *Little* and *Medium* nodes. The fact that the minimum threshold for *Big* has not been updated at this stage is not relevant since *Big* will not be part of the computed combination.

A last phase, performed at the end of Algorithm 4, checks if all architectures are utilized in the BML combination. If there exists an architecture i whose $minThreshold^i$ is greater than or equal to its ${perf_{max}}^i$, it means that the utilization range of this architecture is empty, and thus it must be removed from the infrastructure. Under such case, algorithms 3 and 4 should be executed again with the updated list of BML candidates.

Algorithm 4 $CrossPoints_{Step2}$: Finds crossing points between architectures and combinations of small architectures

Input: BML candidates: BML, Crossing points: $crossPoints$
Output: Updated BML architectures: BML

```
for i ∈ [1, crossPoints.length − 1] do
    current ← LMB[i]
    next ← LMB[i + 1]
    if (crossPoints[i] − 1 % perf_max^current) == 0 then
        j ← crossPoints[i]
        baseLevel ← crossPoints[i] − 1
        comb, power ← idealBML(j − baseLevel)
        while j ⩽ perf_max^next and
            power < powerFor^next(j − baseLevel) do
            j ← j + 1
            comb, power ← idealBML(j − baseLevel)
        end while
        minThreshold^next ← j
    end if
end for
```

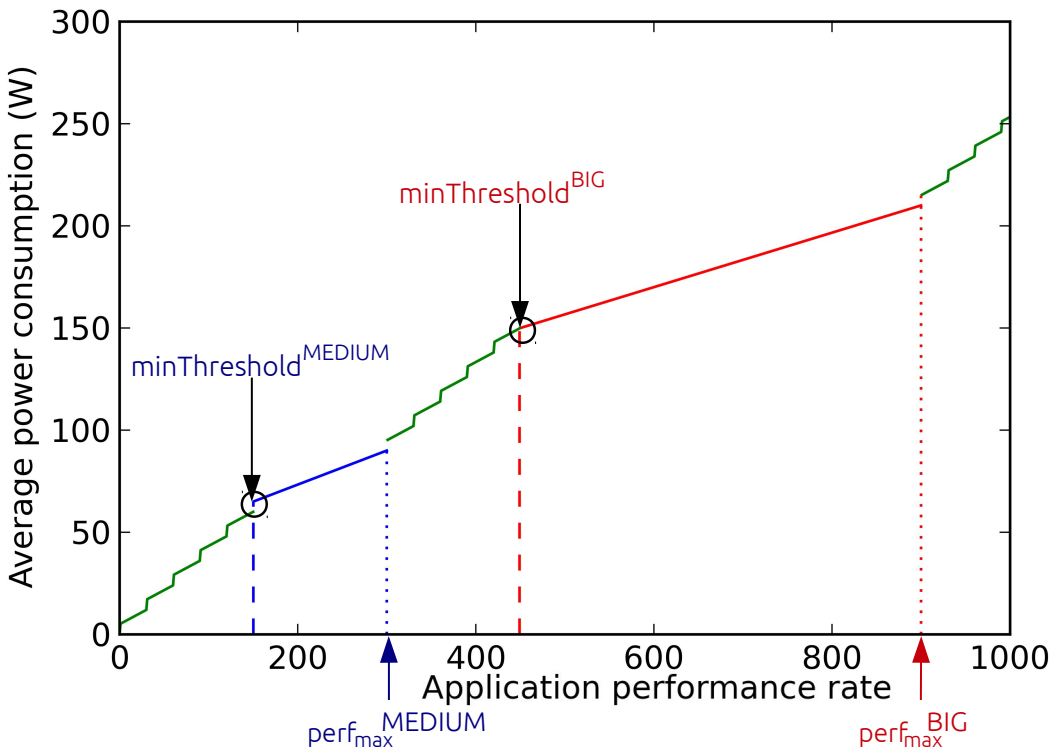


Figure 4.5: BML combination after second step crossing points between Little and Medium, and between combinations of Medium-Little and Big

4.3.5 Final step: Computing Ideal BML Combination

Algorithm 5 details the function that computes the ideal machine combination and its corresponding power consumption to achieve a given performance rate. The building of the BML combination is similar to a bin-packing problem where the architectures and their maximum performance rates represent bins of different sizes. The singularity of our problem is that there is only one object to pack, i.e., the target performance rate, but it can be divided into as many pieces as necessary, and of any size considering a certain unit. The cost to minimize is the power consumption. Steps 2 to 4 sort the bins by size and cost, and determine their minimum utilization thresholds that minimize the cost. What is left, and is performed by this final step, is to divide the amount of performance into several pieces that can fill the bins. In a first stage, we consider the architectures sorted from *Big* to *Little* and seek to fill completely *Big* nodes, then *Medium* nodes, and so on. Architectures are the most energy efficient when running at their maximum performance. In a second stage, we use the minimum utilization thresholds previously computed in order to determine which architectures to choose for achieving the remaining performance rate.

Algorithm 5 $idealBML$: Computes ideal BML combination and its instantaneous power consumption for $perfRate$

Input: Sorted BML list: BML, Performance rate: $perfRate$
Output: BML combination: $combination$, Combination's power consumption: $power$

1: $combination \leftarrow [\ ]$
2: $power \leftarrow 0$
3: $remainingRate \leftarrow perfRate$
4: **for** $arch \in BML$ **do**
5: $\quad nbNodes \leftarrow \lfloor\ remainingRate\ /\ perf_{max}{}^{arch}\ \rfloor$
6: $\quad remainingRate \leftarrow remainingRate - (\ nbNodes \times perf_{max}{}^{arch}\ )$
7: $\quad power \leftarrow power + (\ nbNodes \times power_{max}{}^{arch}\ )$
8: $\quad$ **if** $remainingRate \geqslant minThreshold^{arch}$ **then**
9: $\quad\quad nbNodes \leftarrow nbNodes + 1$
10: $\quad\quad power \leftarrow power + powerFor^{arch}(remainingRate)$
11: $\quad\quad remainingRate \leftarrow 0$
12: $\quad$ **end if**
13: $\quad combination.append(nbNodes)$
14: **end for**
15: **return** $combination$, $power$

4.4 Implementation with Existing Hardware

In this section, we detail our implementation of the BML framework concerning the infrastructure and the application. The following profiling results and BML combination are used in the next chapters for evaluations of the proposed scheduling and resource management policies.

4.4.1 Chosen Hardware

We have broaden the range of hardware for these experiments because we want to be able to make the best choices for building the most energy proportional infrastructure. We keep two of the three machines previously described in Table 3.2 of Chapter 3, which are *Taurus (x86 Intel Xeon)* and *Samsung Chromebook (ARM Cortex-A15)*. To complete the selection we add the *Raspberry Pi 2 Model B*, a credit card-sized board computer containing another ARM processor, the *ARM Cortex-A7*, that consumes less than the *Chromebook* processor. We complete the range of x86 architectures with two other servers, avaible in *Grid'5000* [10] testbed, that all have different type of *Intel Xeon* processors. External wattmeters are utilized to monitor the energy consumption of all devices and servers, fetching one power value per second. Table 4.1 summarizes the characteristics of the chosen hardware.

Codename	Fullname	Processor	Nb cores
Paravance	Dell PowerEdge R630	x86 Intel Xeon E5-2630v3	2 x 8
Taurus	Dell PowerEdge R720	x86 Intel Xeon E5-2630	2 x 6
Graphene	Carri System CS-5393B	x86 Intel Xeon X3440	1 x 4
Chromebook	Samsung Chromebook	ARM Cortex-A15	1 x 2
Raspberry	Raspberry Pi 2 Model B	ARM Cortex-A7	1 x 4

Table 4.1: Summary of selected hardware and their characteristics

4.4.2 Application Choice and Setup

Following the conclusions made in Chapter 3, we choose a **stateless web server** as use-case application. A web server is a perfect example of application whose load varies over time. Its performance is easily characterized with a unique application metric which is the number of requests processed per second. It is malleable as multiple instances of the web servers can be deployed on several machines, and a load balancer allows to distribute the load among these instances. The fact that the web server is stateless eases the migration process as it would just consist in stopping a server instance on the source and launching a new one on the destination machine, and then updating the load balancer. It also facilitates dealing with heterogeneous computing resources as a dedicated version of the web server can be compiled for each different architecture.

Regarding software implementation, we use *lighttpd* [34] as web server. It is an open source solution that is easily available with *Debian* based environments, which exist both for ARM and x86 architectures. The content of the web server is a *cgi* script written in *Python*. Each request consists in a loop of random number generation. To craft requests

of heterogeneous sizes, the number of loop iterations is also chosen randomly between 1000 and 2000. A response to a request is formed by a static *html* page that contains the integer representing the number of iterations.

4.4.3 Profiling Phase (Step 1)

Application Performance and Power Consumption

We choose Siege [54] as web benchmark tool. It simulates multiple clients running in parallel that will access the given web server during a specified amount of time. At the end of the benchmark run, it gives some interesting outputs such as the maximum number of requests processed per second, the average response time, average amount of data transferred, and so on.

Our objective in this profiling phase is to get the maximum number of requests processed in one second for each type of machine, as well as the power consumption associated with it. To do so, we execute the benchmark with an increasing number of concurrent clients. At one point, the maximum number of requests per second stagnates, and the latency starts increasing, that is what we consider as the maximum requests rate for a guaranteed quality of service. Each benchmarking test runs for 30 seconds, and the maximum performance level is computed as the average of 5 benchmark results.

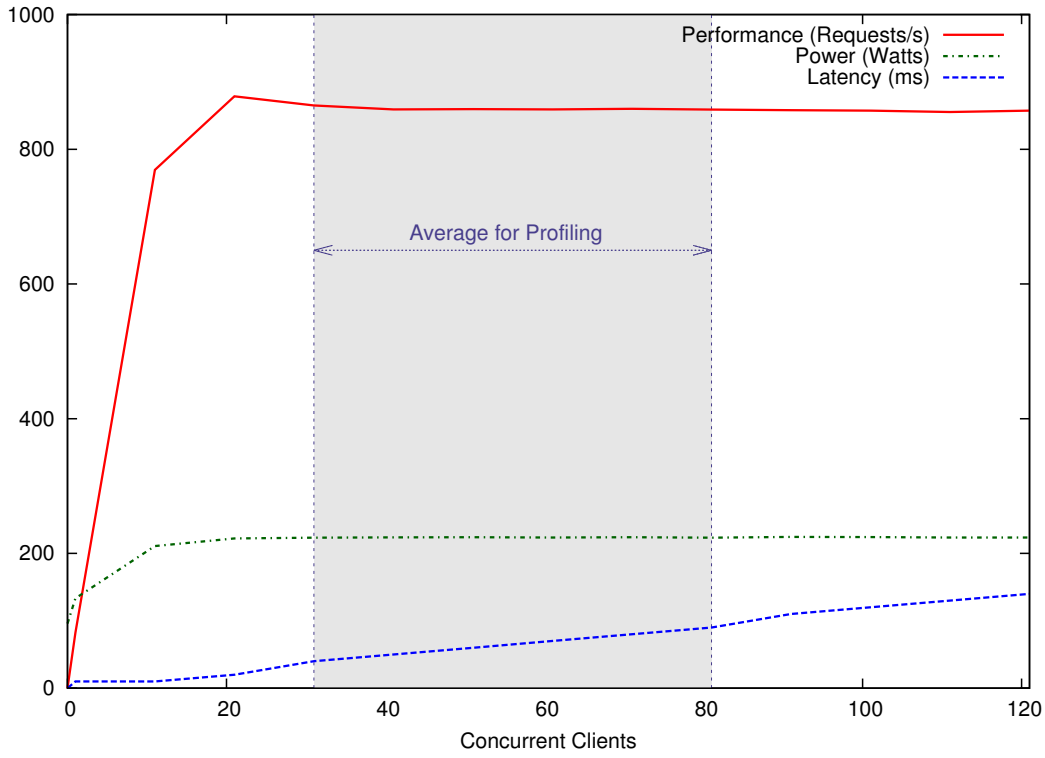


Figure 4.6: Performance, quality of service and power profiles of lighttpd web server running on Taurus (x86 Intel Xeon) for different number of concurrent clients

Figure 4.6 pictures this profiling phase for *Taurus* server. Five benchmark runs are selected as representative executions and their results are averaged to compute maximum performance and its associated power consumption. The first run selected is the one just after the peak performance result, and all the the other selected executions have a latency lower than a certain threshold, in our case 100 ms. For this architecture, the rate of processed requests stabilizes around 860 requests per second. We can see that since this value is reached, the latency is continuously increasing. The power consumption stabilizes around 223 Watts when delivering this level of performance.

We repeat this same profiling methodology for all the chosen architectures. The results are presented in Table 4.2 and the profiles are plotted in Figure 4.8.

Architecture	$perf_{max}$ (Requests/s)	$power_{max}$ (Watts)	$power_{idle}$ (Watts)
Paravance	1331	200.5	69.9
Taurus	860	223.7	95.8
Graphene	272	123.8	47.7
Chromebook	33	7.6	4
Raspberry	9	3.7	3.1

Table 4.2: Performance and power profiles of each architecture.

Switch On/Off Duration and Energy Consumption

We evaluate the time to switch on and off each machine type, as well as the energy consumed during these actions. Figure 4.7 represents the evolution of the power consumption of *Paravance* server during successive switch on and switch off actions. In this experiment, the switch on process lasts 191 seconds until the machine is effectively on and can answer to a *ping* request. After staying 20 seconds idle, the server is powered off, which takes 10 seconds in this case.

We repeat five times each action on each machine type to get average results to constitute the profiles. We compute the total energy consumption for each action in Joules by summing up the acquired power data. Table 4.3 presents these results. The minimum switching intervals T_s computed for all architectures are presented in last column of Table 4.3.

Architecture	t_{On} (s)	e_{On} (J)	t_{Off} (s)	e_{Off} (J)	T_s (s)
Paravance	189	21341	10	657	315
Taurus	164	20628	11	1173	228
Graphene	71	4940	16	760	119
Chromebook	12	49.3	21	77.6	33
Raspberry	16	40.5	14	36.2	30

Table 4.3: On/Off duration and energy consumption, and minimum switching intervals of each architecture

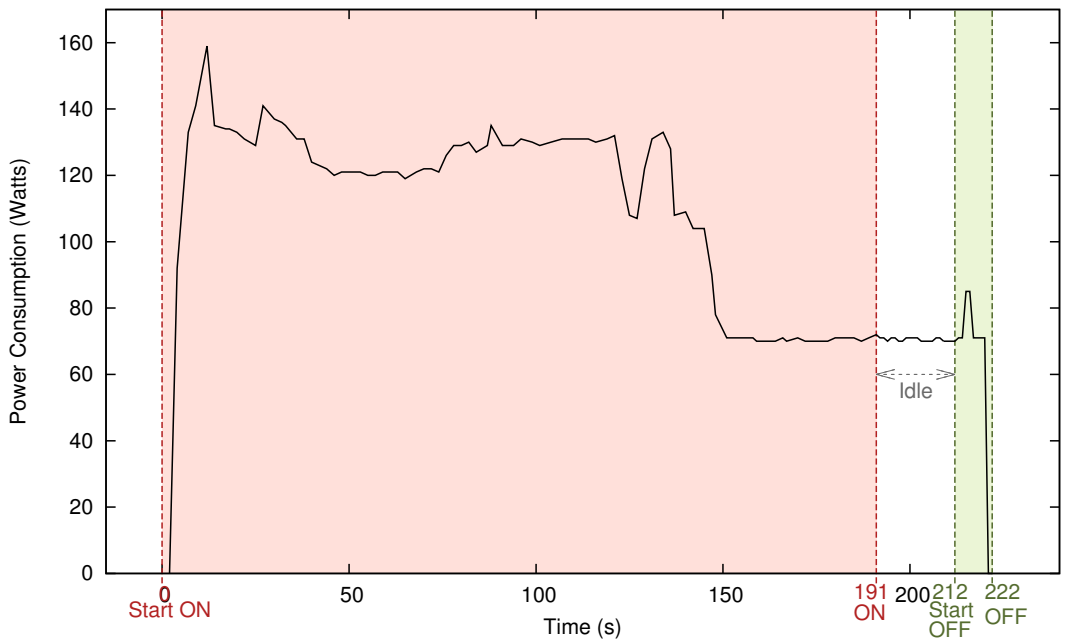


Figure 4.7: Power consumption of *Paravance* server during switch on and switch off actions

4.4.4 Combining Phase (Steps 2 to 4)

Figure 4.8 shows the profiles for the five chosen machines, which correspond to the result of *Step 1*. All curves are unlimited because they are the repetition of the experimental profile to picture multiple machines of the same architecture.

The execution of *Step 2*, which consists in sorting and tagging architectures according to their maximum performance and power consumption, results in the removal of the *Taurus* server. Indeed, its maximum power consumption is higher than *Paravance*'s (223.7 W against 200.5 W) while delivering lower performance (860 requests/sec against 1331). The four remaining architectures are tagged as follows: *Paravance* ← *Big*, *Graphene* ← $Medium1$, *Chromebook* ← $Medium2$, *Raspberry* ← *Little*. *Step 3*, which computes the crossing points between architectures, reveals that the profile of *Graphene* ($Medium1$) never crosses any other architecture's profile. Consequently, it is removed from the list of candidates architectures as it does not help increasing energy proportionality.

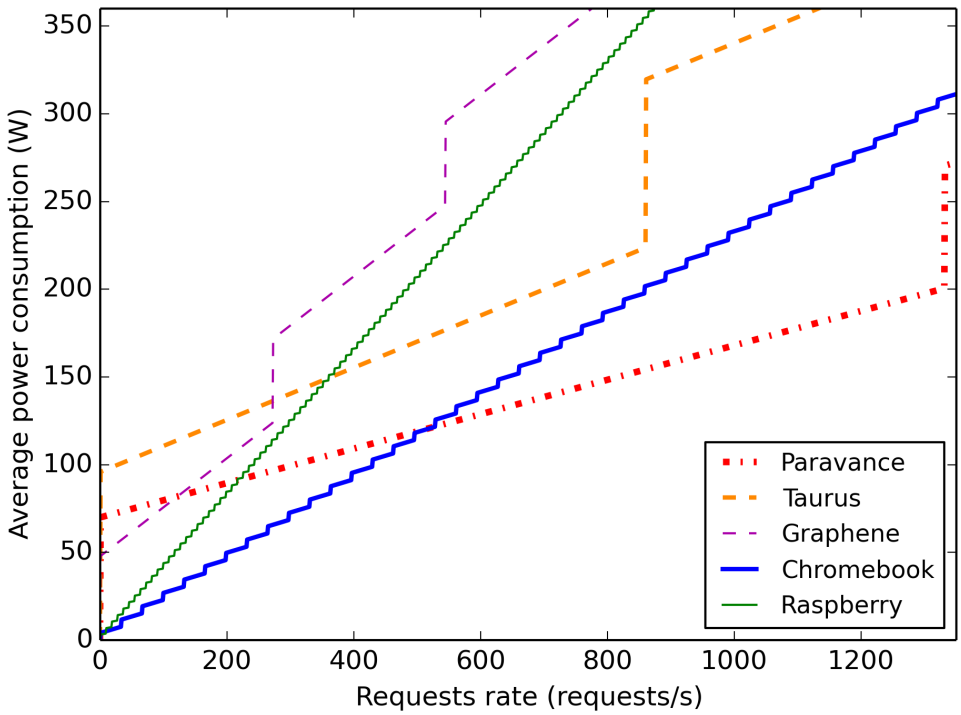


Figure 4.8: Power and performance profiles of web servers acquired from experiments of five different architectures

4.4.5 BML Ideal Combination (Final Step)

Our final heterogeneous infrastructure comprises three types of machines: *Paravance* (*Big*), *Chromebook* ($Medium$) and *Raspberry* (*Little*). Their minimum utilization thresholds are respectively 1 request per second (requests/s) for *Little*, 10 requests/s for $Medium$ and 529 requests/s for *Big*. Additionally, we note $maxNb^i$ the maximum number of machines of each architecture i that compose ideal combinations. For example, it pictures the fact that it is not ideally energy efficient to use more $maxNb^{Medium}$ machines, but instead use machines of type *Big*. In our infrastructure, $maxNb^{Little}$ is 1, $maxNb^{Medium}$

is 16 and $maxNb^{Big}$ has no sense and is set to ∞. In fact, as many *Big* nodes can be utilized, it only depends on the maximum performance needed, or on the maximum number of available *Big* machines in case of an existing infrastructure.

Energy Proportionality at Server Scale

The ideal BML combination, result of *Final Step*, is depicted in Figure 4.9. *Big* architecture's profile is also represented in order to demonstrate the gains of the heterogeneous combination at the server scale. Indeed, the scale of Figure 4.9 is delimited by maximum performance and maximum energy consumption of a single *Big* server. In addition, we introduce a theoretical architecture termed *BML linear*, whose idle power is equal to *Little*'s and maximum power and performance is equal to *Big*'s. It represents here an achievable goal, and allows to show how our solution approaches it. A perfectly proportional architecture would be very close to *BML linear*, the only difference would be its idle power consumption equal to 0.

We evaluate the energy proportionality of our BML infrastructure at the server scale by computing the metrics introduced in 2.1.2. We compute the metrics of our *BML architecture* which is the resulting from machines combination, therefore the comparison with individual machines is a little biased because we do not consider the switch on and off overheads of the different machines of the combination but only considered instantaneous energy consumption.

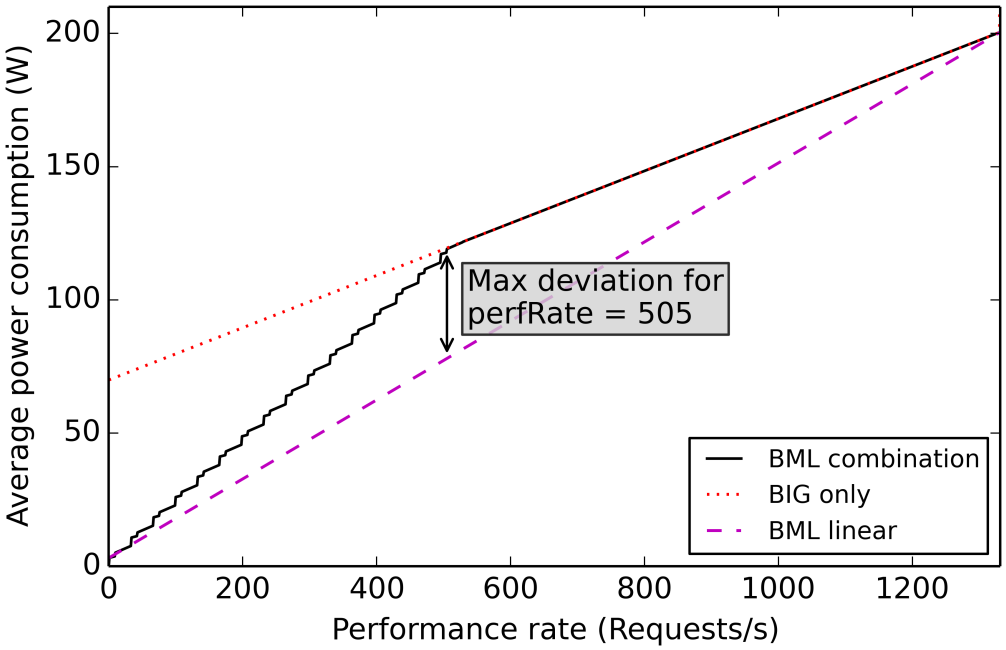


Figure 4.9: Consumption of BML combination over an increasing performance rate, until ${perf_{max}}^{Big}$, compared to *Big* and *BML linear* (BML combination is composed of 1 *Big*, 16 *Medium* and 1 *Little* machines.)

METRIC	*EP*	*IPR*	*DR*	*LDR*	*LD*
Ideal	**1**	**0**	**1**	**0**	**0**
Big (Paravance)	0,6508	0,3486	0,6513	0	0
BML Combination	0,7830	0,0154	0,9845	0,5268	0,1983

Table 4.4: Energy Proportionality metrics computed for BML combination and *Big (Paravance)* server.

Table 4.4 gathers the metrics values for the BML combination and the *Big* server, while the "Ideal" line recalls what would be the values for a perfectly proportional server.

EP characterizes the overall energy efficiency, and those results concludes that BML combination is 20,3% more energy proportional than the *Big* server (EP is 0,7830 against 0,6508).

The two metrics *IPR* and *DR* evaluate the dynamic energy consumption range, or Idle-to-Peak ratio. Even if these two metrics characterize a similar aspect, their formulas differ a lot, as well as their results: *IPR* of BML combination is −95,5% lower than *Big*'s whereas its *DR* is 51,1% greater.

As *LDR* and *LD* metrics are concerned, their values for *Big* are both 0 because we made the assumption of linear power consumption for all our profiled hardware. Consequently, the results for BML combination are necessarily worse than those of the *Big* server. *LD* pictures the overall deviation from linearity (0,1983) while *LDR* focus only on the maximum deviation (0,5268), which explain their different results. The maximum linear deviation is indicated on Figure 4.9 .

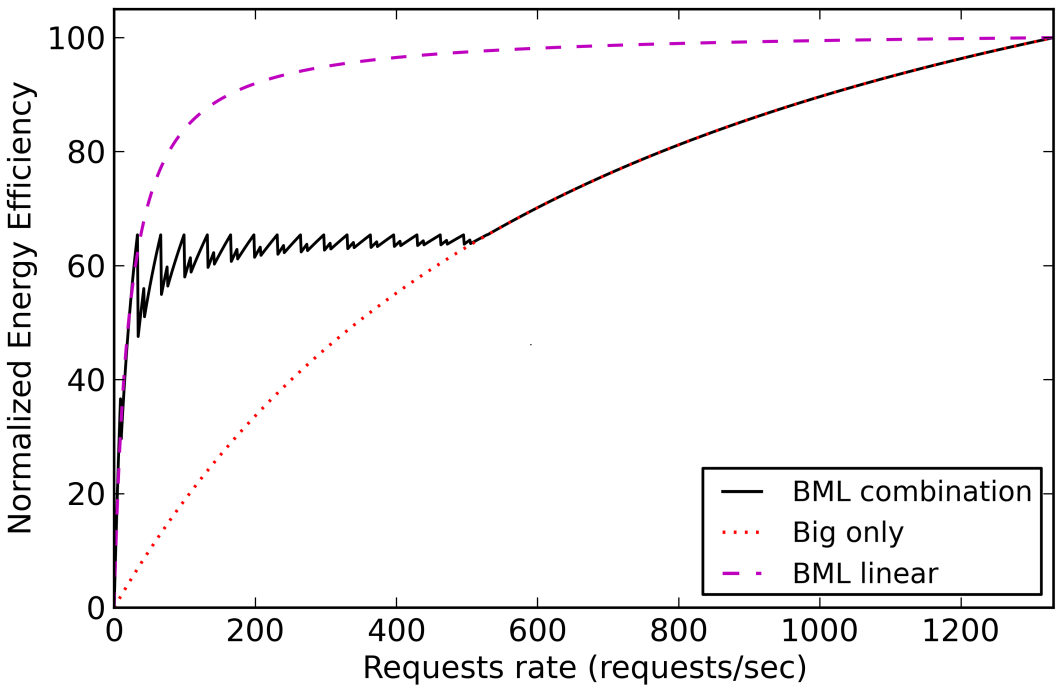


Figure 4.10: Normalized Energy Efficiency of BML combination compared to *Big* and *BML linear*

Another perspective of the BML combination is provided in Figure 4.10 by presenting its energy efficiency, and comparing it to the efficiencies of Big and $BML\ linear$. Energy efficiency is defined as Utilization divided by Power consumption [9]. In our case, utilization is expressed as request rate, ${perf_{max}}^{Big}$ considering 100% utilization. The energy efficiency is normalized considering that (${perf_{max}}^{Big}$,${power_{max}}^{Big}$) represents maximum energy efficiency. The BML combination improves a lot energy efficiency compared to a single Big machine, especially for the first third of utilization range, being around 60% efficiency.

Through this chapter we have explained the BML framework in details and given an implementation with real hardware, as well as its energy proportionality evaluation at server scale. The following Chapters 5 and 6 present two different provisioning algorithms and evaluate their relevance and efficiency facing real workload traces. We demonstrate the energy proportionality gains of our BML infrastructure compared to homogeneous solutions, while taking all resources reconfigurations into account.

– Chapter 5 –

Ideal BML Algorithm

This chapter presents our first algorithm designed for provisioning BML infrastructure. We also introduce our simulator that we used to evaluate multiple algorithm settings for different scenarios with both synthetic and real workload traces.

5.1 General Functioning of the Algorithm

This first scheduling algorithm is called *Ideal BML* as it is entirely based on the ideal combinations of machines pre-computed as detailed in Chapter 4. These combinations are said *ideal* because they constitute the most energy proportional infrastructure with the considered set of architectures.

The general functioning of the algorithm is the following:

A load value is selected as the future level of performance needed. In our work, we are considering a perfect knowledge of the future workload. We emulate a load prediction mechanism by considering a *sliding look-ahead window* over the future load values. Two approaches are used to determine the predicted load: either considering the average of the window values; or picking the maximum value. We discuss the results obtained with these two alternatives as well as the length of the look-ahead window in section 5.3.

The ideal machines combination for providing this future level of performance is computed. If this ideal combination is different from the current configuration, a reconfiguration towards the ideal combination is performed. During the reconfiguration, no other decision can be taken. This ensures the completion of switch on and off actions before a new reconfiguration decision is made.

The next window used to predict the load starts from the reconfiguration completion time. When the prediction results in no changes in hardware combination, the window just slides one time step forwards, which is a second in our case.

5.2 BML Simulator

To evaluate our infrastructure with different scheduling approaches without any limitations in terms of hardware installation, we have developed a simulator. Thanks to experimental profiling, we have all the necessary parameters to compute the energy consumption for different scenarios of data centers hosting stateless web servers.

The simulator, coded in *Python*, takes as inputs the experimental hardware profiles, each of them being described in a *Json* file. It contains all the algorithms described in section 4.3 allowing to sort architectures and build the BML combinations. To run a simulation, it needs a file that describes the evolution of the application workload over time. Then, at each time step, it knows the actual application load and can compute the energy consumed to process the requests by the currently powered on machines, and eventually take reconfiguration decisions such as switching some nodes on or off.

If requests that arrive at a time step cannot be processed immediately, due to resource under-provisioning, they are put in a simulated web server waiting queue. Then, at each time step, the already waiting requests take priority over requests that have just arrived. To avoid starvation, requests wait for 2 seconds at maximum before being discarded.

All parameters characterizing the scenario, as well as many parameters describing the execution of the simulation are collected and exported as output in a *Json* formated file. This allows to easily extract data for evaluations once the simulations are completed, and to produce graphs that represent the temporal evolution of the simulation.

Among other metrics, we compute the percentage of processed requests among arriving requests, and the percentage of delayed and lost ones. We also define the metric *JpR* for *Joules per Request* which is computed as the total energy consumed during the run over the total number of processed requests, to quantify the energy efficiency:

$$JpR = \frac{total\ energy\ consumed}{total\ processed\ requests} \tag{5.1}$$

The percentage of utilization of the infrastructure represents the number of processed requests over the processing capacity of the current infrastructure's configuration. We express it as the average of all utilizations computed for each second:

$$Utilization_{PerSecond} = \frac{processed\ requests}{capacity\ of\ infrastructure} \tag{5.2}$$

As workload traces for our simulations, we use the 1998 World Cup website access logs (available at [1]). It contains the access records to the World Cup web site. Traces have been collected during a period of 86 days, between April and July 1998. Total received number of requests is over one billion [5]. Figure 5.1 plots the maximum and average requests rate, expressed in number of requests per second, for each day of the World Cup log traces. We can see with the standard deviation that the daily variation is very important. For instance if we focus on the highest peak, which was on the June 23rd, about 4000 requests per second, the average request rate on this day was approximately of 900 requests per second. We have selected these traces as they contain higly variable load over time and are representative of the use case we are targetting.

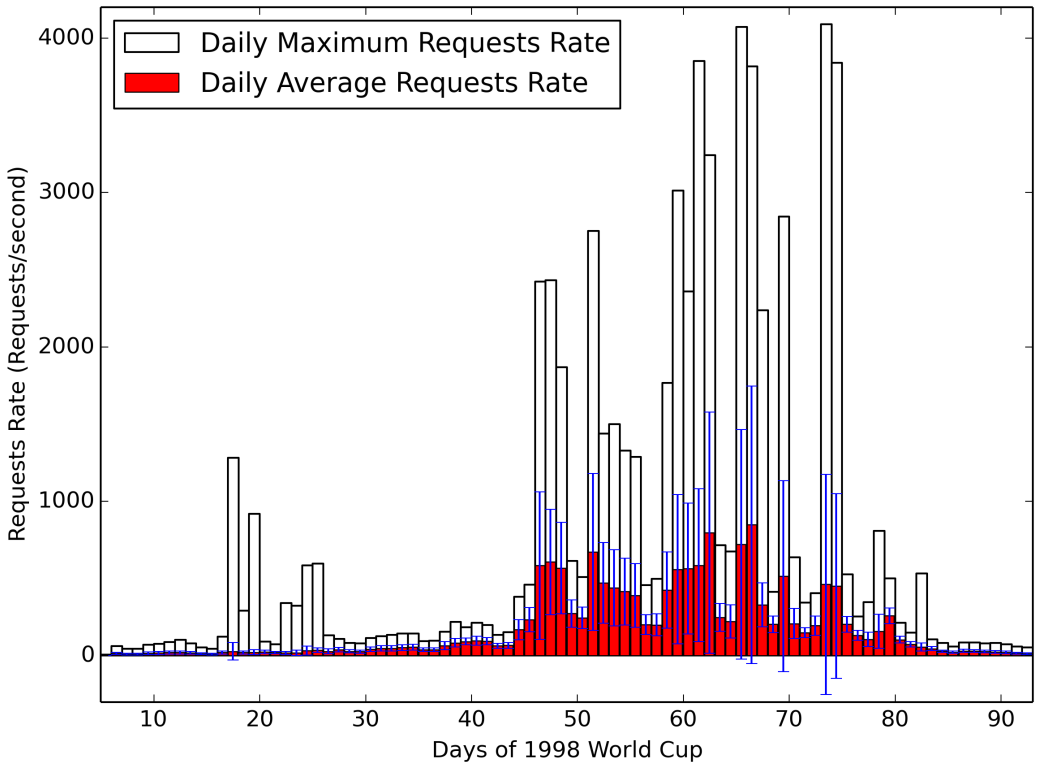


Figure 5.1: Maximum and average requests rates for all days of 98 World Cup traces

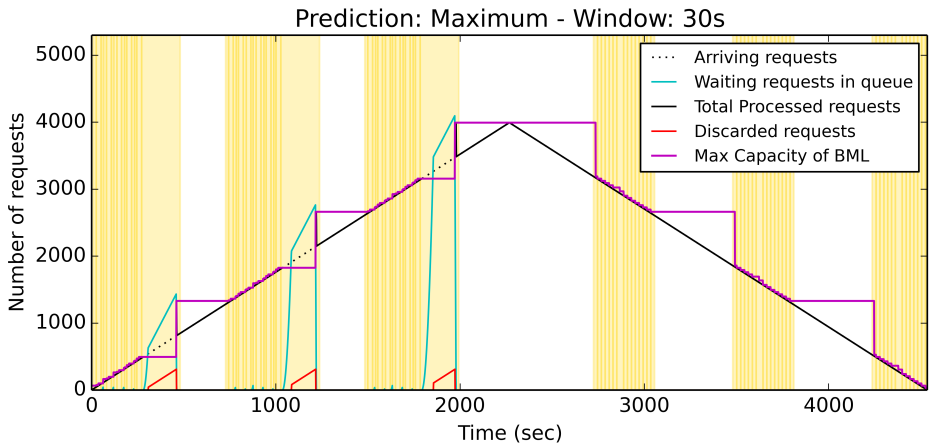
5.3 Study on Prediction Types and Window Sizes

The algorithm takes reconfiguration decisions based on future load knowledge. In this section, we study how the future load value should be selected, meaning how far in the future should we look, and which level of performance should be considered. We choose to test two different types of prediction: (i) selecting the average value of the look-ahead window; and (ii) considering the maximum value. Regarding the length of sliding window, we also select two alternatives: (i) a short window of 30 seconds; and (ii) a long window of 315 seconds. These numbers correspond to the minimum switching intervals T_s^i, 30s being the shortest of them, associated to the *Little* architecture, and 315s the longest, for the *Big* architecture. The minimum switching interval of the *Medium* architecture is 33 seconds, and thus very close to *Little*'s, that is why we only choose one of them for simplification as they both give similar results. Theses choices translate into four different settings for future load selection.

To understand the behavior of the scheduling algorithm and be able to easily draw conclusions, a simple generated trace is used for simulations. It consists in one upward and one downward phase of the same length, starting from a request rate of 0 to a maximum request rate of 3993 requests/second, corresponding to 3 times the maximum processing capacity of *Big* architecture.

Period	%Process.	%Delay.	%Lost	JpR	%Util.	NbRec
Up	98.3	18.3	1.7	0.1783	93.5	49
Down	100	0	0	0.1638	85.9	44
Total	**99.2**	**9.1**	**0.8**	**0.1710**	**89.7**	**93**

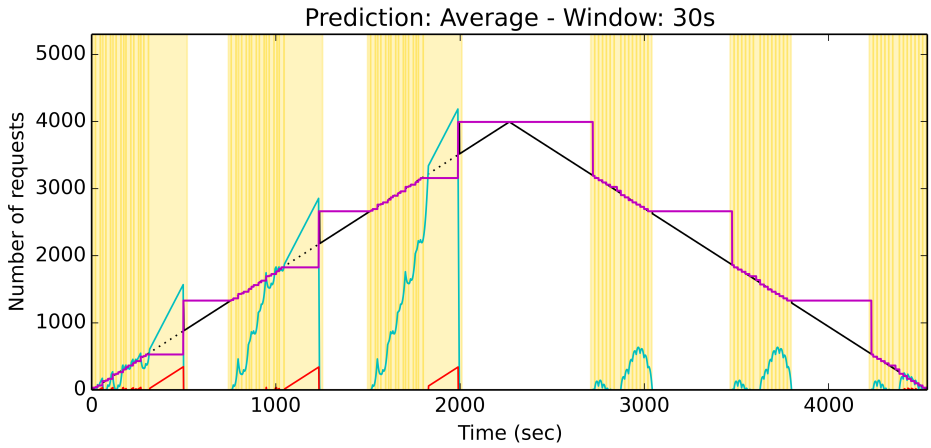
(a) Evaluation Metrics for Prediction: Maximum - Window: 30s



(b) Temporal Evolution for Prediction: Maximum - Window: 30s

Period	%Process.	%Delay.	%Lost	JpR	%Util.	NbRec
Up	97.8	38.5	2.2	0.1782	95.9	51
Down	99.99	3.8	0.01	0.1632	88.4	45
Total	**98.9**	**21**	**1.1**	**0.1706**	**92.1**	**96**

(c) Evaluation Metrics for Prediction: Average - Window: 30s

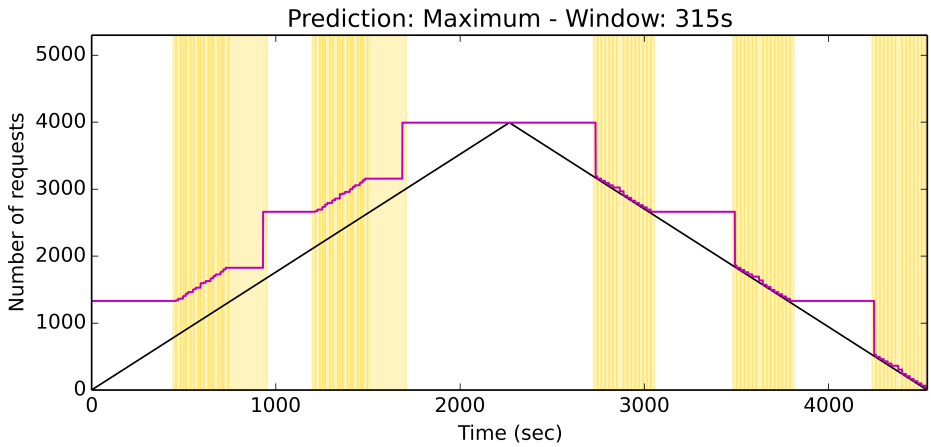


(d) Temporal Evolution for Prediction: Average - Window: 30s

Figure 5.2: Comparison of different prediction settings and same short window length on an input trace consisting of regular upward and downward slopes.

Period	%Process.	%Delay.	%Lost	JpR	%Util.	NbRec
Up	100	0	0	0.1822	70.1	34
Down	100	0	0	0.1638	85.9	44
Total	**100**	**0**	**0**	**0.1730**	**78**	**78**

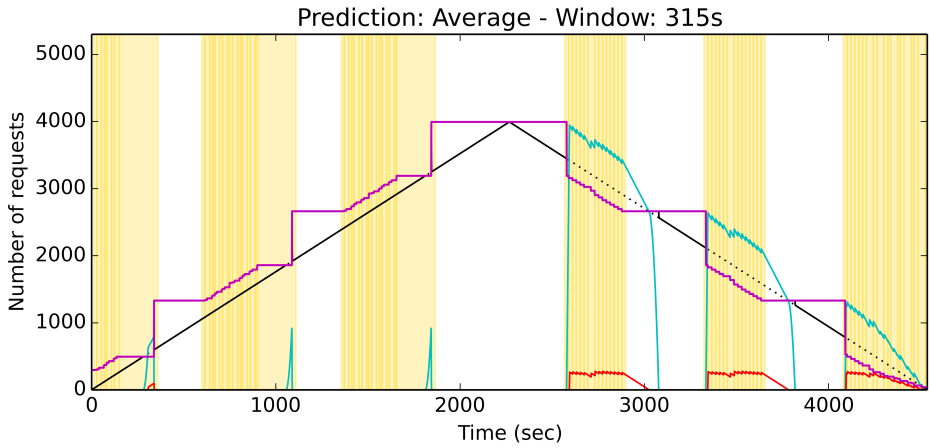
(a) Evaluation Metrics for Prediction: Maximum - Window: 315s



(b) Temporal Evolution for Prediction: Maximum - Window: 315s

Period	%Process.	%Delay.	%Lost	JpR	%Util.	NbRec
Up	99.95	0.9	0.05	0.1822	81.9	42
Down	94.5	50.3	5.5	0.1609	94.5	49
Total	**97.2**	**24.9**	**2.8**	**0.1718**	**88.2**	**91**

(c) Evaluation Metrics for Prediction: Average - Window: 315s



(d) Temporal Evolution for Prediction: Average - Window: 315s

Figure 5.3: Comparison of different prediction settings and same long window length on an input trace consisting of regular upward and downward slopes.

Figures 5.2 and 5.3 present the simulation results for the four different settings. For each simulation it shows a table with evaluation metrics: percentage of total processed requests, percentage of requests processed with a delay of 1 or 2 seconds, percentage of discarded requests, joules consumed per processed request (JpR), infrastructure utilization and the number of reconfiguration decisions. These metrics are computed separately for the upward and downward phases (from 0 to 2267s and from 2268 to 4536s), as well as for the total length of the simulation. Each graph represents the temporal evolution of the simulation: arriving requests, maximum processing capacity of the BML combination, waiting requests and discarded requests over time. The vertical lines correspond to the periods when a reconfiguration is on-going.

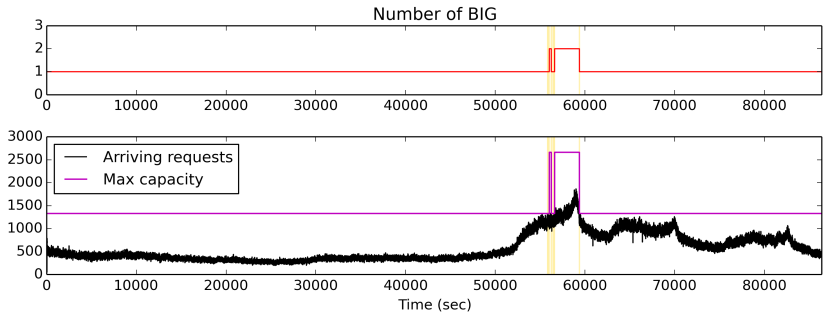
We can clearly see that the system behaves differently during upward and downward phases. This is due to the different temporalities of switch on and switch off actions. Indeed, when a decision of powering on one or several machines is taken, it takes some time until the machines are effectively on and running, that is why the long window gives better results during the upward period. On the opposite, powering down one or several machines has an immediate action because as soon as the machines start their shutdown process, they are not available for processing anymore. Consequently, the short window is more accurate for the decreasing period.

The only setting leading to zero lost and zero delayed requests is the maximum value over the long window. Regarding energy consumption, it achieves a *joules per request* metric (JpR) of 0.1730. It is the best choice for applications that are *critical* and have strong QoS requirements. If the applications are more *tolerant*, then the choice is less constrained. Both settings with short windows provide similar results: 0.1710 and 0.1706 JpR and only 0.8% and 1.1% of discarded requests for respectively a window size of 30s and 315s. For all the following results concerning the Ideal BML algorithm, we choose the setting of the maximum value over the long window as it allows to process all the incoming requests without delay nor loss.

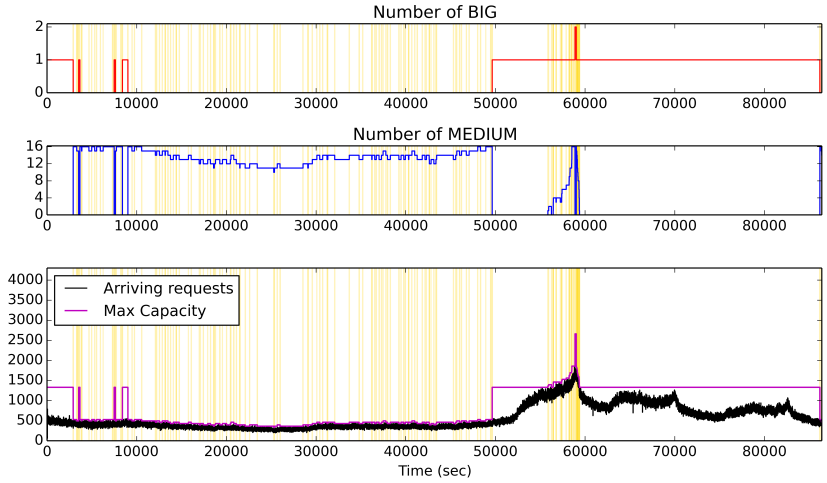
5.4 Big Only, Big-Medium or BML?

To evaluate the gains brought by heterogeneity, we compared the BML infrastructure against other configurations: a homogeneous infrastructure composed only of *Big* machines, and a heterogeneous infrastructure but composed of both *Big (Paravance)* and *Medium (Chromebook)* nodes.

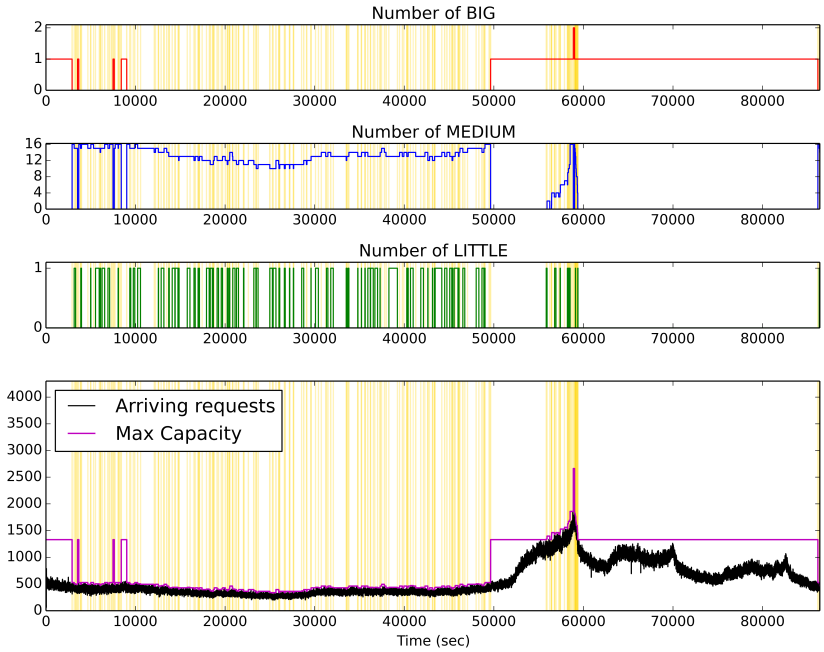
We run the simulations for the three different scenarios with the traces from the 1998 World Cup [1]. We focus on the 48th day as it is one comprising the largest number of requests: the average request rate is about 566 requests per second, with a peak demand at 1867 requests per second. The setting is a prediction based on the maximum value on a long window of 315 seconds, as though the application were critical. This enables us to compare the different results with the same number of total processed requests. Figures 5.4(a), 5.4(b) and 5.4(c) show the temporal evolution of the simulation for respectively the B, BM and BML cases. Apart from the processed requests and the maximum processing capacity of the infrastructure, we also show the number of each type of machines over time, and vertical lines corresponding to reconfiguration periods.



(a) Big only - JpR: **0.2266**, Utilization: **40.8%**, NbReconf: **4**



(b) Big-Medium - JpR: **0.2153**, Utilization: **69.5%**, NbReconf: **146**



(c) Big-Medium-Little - JpR: **0.2152**, Util.: **70.1%**, NbReconf: **221**

Figure 5.4: Comparison of different infrastructure scenarios for Day 48.

Table 5.1: Total energy difference percentages of BML infrastructure compared to Big-Medium and Big only scenarios

Energy Difference	Compared to Big-Medium	Compared to Big only
Best *per day*	-5% % (day 91)	-92.6% (day 8)
Worst *per day*	-0.05% (day 62)	-2% (day 43)
Average *per day*	-1.05%	-57.9%

We see that heterogeneity allows to adapt the infrastructure's maximum capacity at finer grain. It consequently leads to higher utilization; 70.1% for BML, 69.5% for BM against only 40.8% for the Big only infrastructure. The energy waste is reduced by higher utilization, especially during periods of low load as in the 50 000 first seconds of the day. Similar conclusion can be drawn from the joules per request metric: 0.2152 JpR for BML, 0.2153 JpR for BM against 0.2266 JpR in Big only case.

The comparative results of the different scenarios over all the 86 days of the World Cup are shown in Table 5.1. The BML configuration consumes on average -1.05% less energy than the Big-Medium version, the minimum difference being -0.05% and the maximum -5%. In comparison to the homogeneous infrastructure only composed of *Big* nodes, the BML version consumes on average -57.9% less, the maximum difference being -92.6% and the minimum -2%.

The small difference between BML and BM infrastructure stems from the fact that our BML combination is not optimal. Indeed, the *Little* architecture has a small utilization range of only 9 requests. This explains why its presence in the infrastructure cannot bring significant improvements. Nevertheless, we observe that in the BML infrastructure, there is a higher number of reconfigurations than in the BM case, exactly 75 reconfigurations more, while the results in terms of joules per request are better. It means that we still benefit from a more heterogeneous platform even if the utilization range of the additional architecture is small.

Ideally, it would be preferable for each architecture to have the same range of utilization to improve the results significantly.

5.5 Comparison with Lower and Upper Bounds

In this section we evaluate our Ideal BML algorithm to a theoretical BML lower bound, and two homogeneous upper bounds corresponding to existing data center management. We run the simulations for days 6 to 92 of 1998 World Cup traces [1].

The considered scenarios are as follows:

- *Global UpperBound* corresponds to a data center with a constant number of homogeneous *Big* servers during the whole duration of the World Cup, computed according to the maximum request rate. In these traces, the peak rate is *4089* requests per second during day 73. Consequently, the infrastructure contains 4 *Big* machines that are always powered on. This upper bound is an example of a classical over-provisioned data center.

- *Per Day UpperBound* pictures a data center composed of homogeneous *Big* servers. In this scenario, the infrastructure is dimensioned each day according to the daily maximum rate. This is an example of coarse grain capacity planning.

- *Ideal BML* is our BML infrastructure and the Ideal BML provisioning algorithm described in this chapter, with the same settings as for previous section: a prediction based on the maximum value over a long look-ahead window of 315 seconds. The total consumption per day contains the energy consumed by computation and the energy from on/off reconfigurations made during the day.

- *Theoretical LowerBound* represents the minimum computing energy achievable with our BML infrastructure if the data center could be dimensioned every second with the ideal BML combination. This is an unreachable lower bound considering no on/off latency and no on/off energy costs. This lower bound pictures also the maximum energy proportionality we could reach with our infrastructure.

Figure 5.5 summarizes the results. It must be interpreted per day as the energy costs of on/off actions between days are not taken into account. Our *Ideal BML* solution is very close to the theoretical lower bound. On average over these 86 days, our Ideal BML algorithm results in a total energy consumption 31% higher than the theoretical lower bound, the minimum being 6.6% for day 52 and the maximum 149.9% for day 23. Difference percentages of Ideal BML energy consumption compared to lower bound and both upper bounds are summarized in Table 5.2. The graph of Figure 5.5 demonstrates the high static costs coming from classical over-provisioned data centers represented by *Global UpperBound*, and allows to see the objective reached with our solution that is an energy consumption more proportional to the actual daily load.

This energy proportionality is clearly noticeable on Figure 5.6, which is a scatter plot of the daily total energy consumption of the infrastructure regarding the daily cumulated number of requests. We observe the important waste of energy in the two homogeneous upper bound cases compared to our dynamically reconfigured heterogeneous infrastructure whose energy consumption follows the load evolution. Moreover, the proximity of our solution with the theoretical lower bound proves the relevance of the infrastructure reconfigurations and that the energy consumed by switching machines on and off does not represent a significant overhead.

Table 5.2: Total energy difference percentages of Ideal BML algorithm compared to lower and upper bounds

Energy Difference	Compared to Lower Bound	Compared to Upper Bound (Day)	Compared to Upper Bound (Global)
Best *per day*	+6.6 % (day 52)	-92.6% (day 8)	-98.1 % (day 8)
Worst *per day*	+149.9 % (day 23)	-11.6 % (day 54)	-55.3 % (day 66)
Average *per day*	+31 %	-65.1 %	-85.9 %

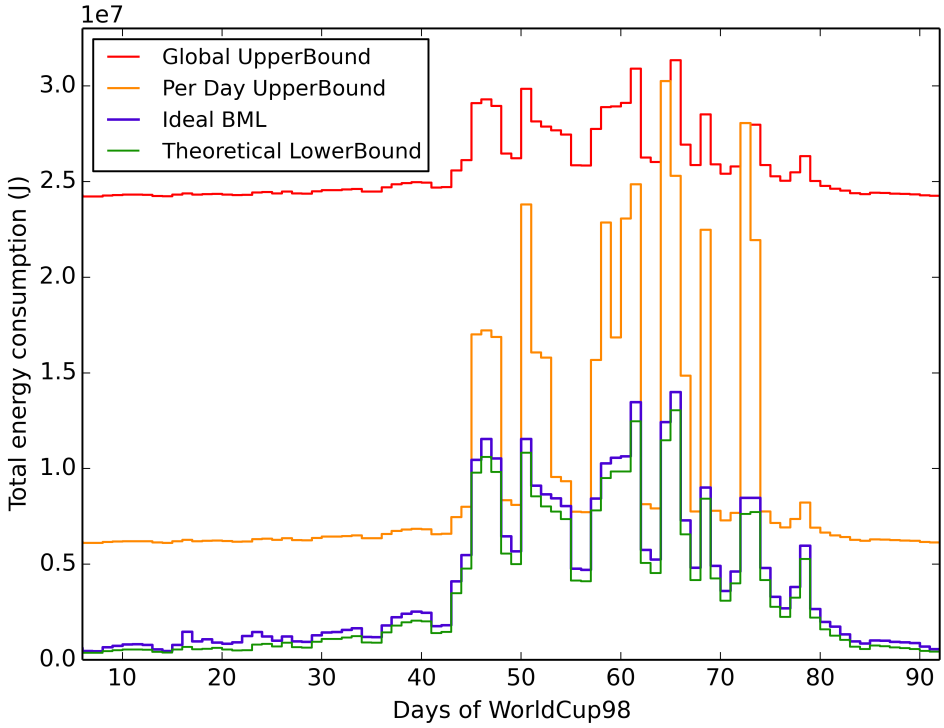


Figure 5.5: Total energy consumption comparison with lower and upper bounds for all days.

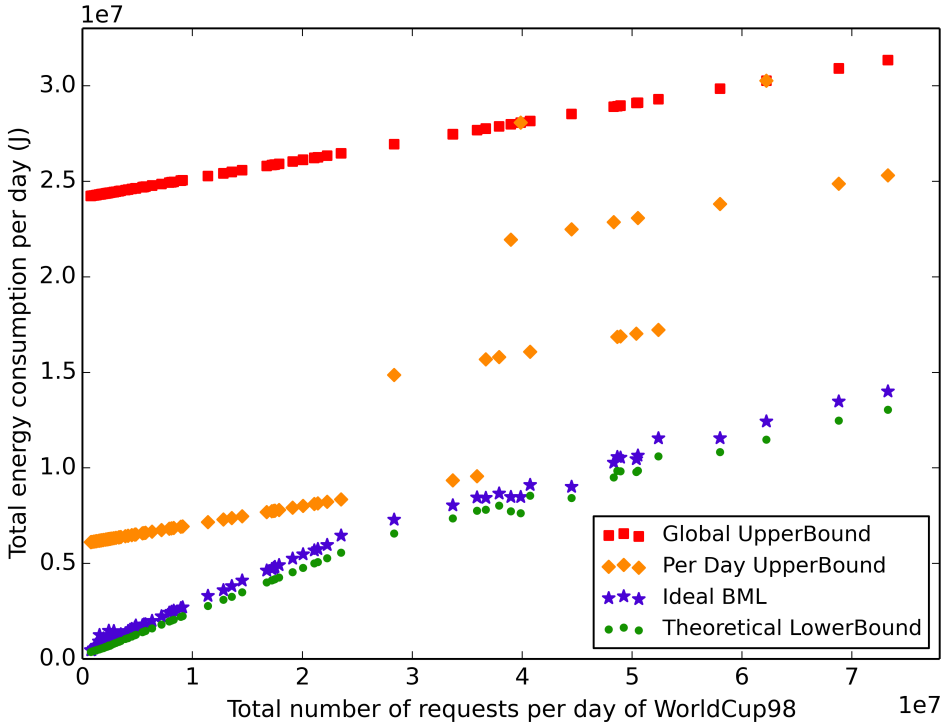


Figure 5.6: Energy proportionality comparison with lower and upper bounds

5.6 Conclusions and Limitations of this Algorithm

Our BML infrastructure combined with this Ideal BML algorithm allows to get closer to energy proportionality, as it drastically reduces the energy consumption compared to homogeneous infrastructures and classical over-provisioned data centers. We also demonstrate that our algorithm provides results close to the theoretical BML lower bound, which is very satisfying as this lower bound does not consider reconfiguration overheads.

However, we discuss that the length of the look-ahead window needs to be long if the application has strict QoS requirements. This can result in slightly over-provisioned configurations. We believe that it should not be necessary to make a compromise and choose only one long window, but that several look-ahead windows can be considered.

Moreover, with Ideal BML we only authorize machine combinations that are part of the pre-computed ideal combinations. Unfortunately, this can incur some heavy reconfigurations that may not be ideal. For example this happens when the load rapidly oscillates between two performance levels that are quite close but correspond to two very different machine combinations. To illustrate, we can imagine two successive ideal combinations (not corresponding to the actual hardware we have presented): combination C_1 composed of 1 *Medium* node and 4 *Little* nodes, and combination C_2, more powerful, composed of 2 *Medium* nodes. If the current configuration is C_1 and the load forecast is a short peak, it could be more efficient to power on an additional *Little* machine for a short amount of time, resulting in a temporary configuration with 5 *Little* nodes that is not part of ideal combinations, than to perform the complex reconfiguration towards C_2 consisting in powering on 1 *Medium* machine and switching off 4 *Little* ones.

We aim at tackling these limitations with the scheduling algorithm described in Chapter 6. It considers multiple look-ahead windows and takes into account reconfigurations overheads in the decision process.

– Chapter 6 –

Multi-Terms Algorithm

In this chapter, we describe the second provisioning algorithm we have designed to deal with the limitations of the first version. We give a comparative evaluation of their results facing the same scenarios. We also discuss some characteristics of the infrastructure that are required for this algorithm to be more efficient.

6.1 Motivations for this Algorithm

We aim at solving the limitations of Ideal BML algorithm by taking some distances with the pre-computed BML combinations and allowing more possible configurations. We no longer think only regarding ideal combinations but consider the switch on and off actions of the different machines separately.

An objective of this algorithm is to take full advantage of the BML heterogeneous infrastructure with all its specifications, and no matter the hardware implementation. Indeed, this algorithm is meant to be totally generic and effective whatever is the number of different architecture types. One limitation of the Ideal BML algorithm is its unique look-ahead window and that its length is forced to be set according to the longest minimum switching interval T_s to make the most relevant reconfigurations decisions. In fact, as we have shown when profiling real hardware in Chapter 4, machines can have very different switch on durations and then very different switching intervals. Therefore this algorithm considers as many look-ahead windows for switch on actions as there are different switch on durations.

In addition, switch on and switch off durations are also very different. We have observed that for most machines, switch off takes less time than switch on. But what is more important to take into account is that switch off actions have an immediate effect on the infrastructure. That is why Ideal BML algorithm and its long look-ahead window can not be the most optimal to decide when to power off machines. In this algorithm, we consider different look-ahead windows for switch off than for switch on actions.

We name this new approach Generic Multi-Terms algorithm to highlight its two most important characteristics: the genericity in terms of considered infrastructure, and the multiple look-ahead windows of different lengths and starting at different times in the future. The functioning of the algorithm is defined in details in the next section.

6.2 Description of the Algorithm

Our Multi-Terms Algorithm is composed of two main categories of actions: (i) load-reactive actions and (ii) energy-saving actions, which put together translate into three different types of actions : *Switch-On*, *Switch-Off*, and *Back-to-Ideal*. The first two consist in actions of machines switch on or off, considered independently, while the last one considers all the powered-on machines as a combination and decides to reconfigure the infrastructure towards a more energy-efficient combination if possible. At each time step, the algorithm starts by proposing load-reactive actions. In case no reactive actions are needed, which means that the capacity of the current infrastructure composition is sufficient to process the incoming workload, the algorithm may propose an energy-saving action. We detail the implementation of all these reconfiguration decisions in the following explanations.

In our work, we assume that the data center operator – executing our dynamic provisioning algorithm – has complete knowledge of the workload ahead of time, and consequently, our algorithm will provision exactly to always satisfy the peak loads. As we want to take advantages of the different temporalities of the reconfigurations actions, the algorithm analyzes the future load knowledge via different look-ahead windows whose sizes are precisely chosen. In our case, we consider the maximum load value of a window to be the future performance rate used for provisioning.

Because our algorithm takes reconfiguration decisions that will take place at different moments in the future, we assume there is a system memorizing the decisions, therefore knowing at any moment what are the ongoing switch-on or off actions. Consequently, it is possible to compute the future processing capacity of the infrastructure knowing the current one and the ongoing actions.

6.2.1 Load-Reactive Actions

At each time step, the algorithm proposes load-reactive actions. It tries to find what switch-on or -off actions are the most appropriate reactions to the incoming workload.

- **Switch-On** actions:

Let denote current time t_{now}. To decide if any switch-on actions are needed for machines of a given architecture i, the considered look-ahead window must start at $t_{now} + {t_{On}}^i$. Indeed, as machine of type i will take ${t_{On}}^i$ seconds before being ready to compute, it is not necessary to look at future load before this point. As we want to avoid over-provisioning as much as possible, we decide to take one time step Δt as window length. For each architecture i, we note $window^i_{short}$ its future load window used to decide switch-on actions.

On Figure 6.1 are pictured the look-ahead windows for switch-on actions for an illustrative example of a data center containing three architecture types noted A, B and C, which would correspond respectively to *Little*, *Medium*, and *Big*. We also consider that their switch on and switch off durations, as well as T_s, follow the same order as their performances. We discuss as the end of this chapter what are the consequences if this

condition is not fulfilled. To ease the notations in the following and avoid repeating t_{now}, we consider that $t_{now} = t_0 = 0$.

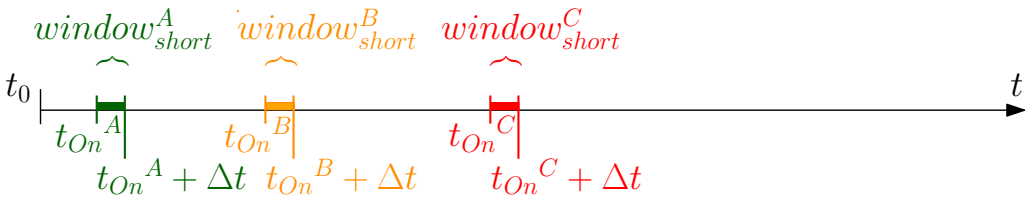


Figure 6.1: Short look-ahead windows used for switch-on actions.

We detail in the following the successive steps for the computation of the quantity of machines to switch on. We assume that we are not limited in the quantity of machines of each type. Because we want to respect the QoS, the algorithm makes the decision to switch on as many machines as needed to answer the predicted future load difference. But in order not to switch on all types of machines and thus results in an over-provisioned architecture, the decision to switch on some machines of architecture i is only taken if the predicted load difference is superior or equal to the minimum utilization threshold $minThresh^i$ of this architecture.

The decision process for Switch-On actions is as follows:

For all architecture types $i \in \mathcal{M}$:

1. Compute the load prediction for $window^i_{short}$.
2. Compute the maximum difference $diff^i_{short}$ between the load prediction and the future capacity of the infrastructure during the considered window.
3. **If** this difference is greater than or equal to the minimum utilization threshold $minThresh^i$, it means that machines of type i need to be switched on;
   Then compute the minimum quantity of machines i necessary to process the predicted load difference:
   $nbOn^i = \lfloor\ diff^i_{short}\ /\ perf^i_{max}\ \rfloor$
4. **Else**, no machine of type i needs to be switched on, $nbOn^i = 0$.

- **Switch-Off** actions:

Switch-off actions are also part of load-reactive actions because we want to shut down all machines becoming unnecessary when the load decreases in order to save energy. We discussed the minimum switching interval T^i_s in the prerequisites, it justifies that the look-ahead windows must start from current time t_{now} and have a length of T^i_s. Figure 6.2 illustrates these switch-off windows for illustrative architectures A, B and C, denoted $window^i_{imm}$ as they begin immediately.

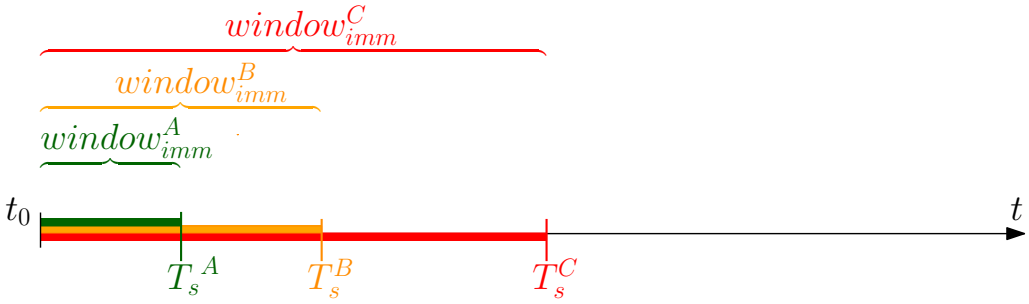


Figure 6.2: Immediate look-ahead windows used for switch-off actions.

Switch-Off actions are decided as follows:

1. **For** all architecture types $i \in \mathcal{M}$ whose current quantity of machines powered on nb^i is positive; Propose switch-off actions:
   (a) Compute the load prediction for $window^i_{imm}$.
   (b) Compute the maximum difference $diff^i_{imm}$ between the load prediction and the future capacity of the infrastructure during the considered window.
   (c) **If** this difference is negative, meaning that the load is decreasing,
   Then compute the maximum quantity of machines of type i that can be turned off:
   $nbOff^i = \min(\lfloor\ |diff^i_{imm}|\ /\ perf^i_{max}\ \rfloor, nb^i_{On})$
   (d) **Else**, no machine of type i can be switched off, $nbOff^i = 0$.
2. Compute all possible switch-off reconfigurations as the combinations of all proposed switch-off actions.
3. Remove the switch-off reconfigurations which do not respect the QoS during their associated $window_{imm}$.
4. Sort the switch-off reconfigurations in the decreasing order of their impact in term of processing capacity.
5. Choose the most appropriate switch-off reconfiguration:
   (a) **For** all switch-off combinations:
   If one of them allows to reconfigure the infrastructure towards an ideal combination for any $window^i_{imm}$,
   Then perform this reconfiguration.
   (b) **Else**, perform the switch-off reconfiguration with the biggest impact in term of processing capacity.

6.2.2 Energy-Saving Actions

- **Back-to-Ideal** actions:

If no load-reactive actions have been proposed in the first phase of the algorithm, this second phase tries to propose an energy-saving action by reconfiguring the infrastructure towards an ideal and energy-efficient combination of machines. In fact, we consider as Back-to-Ideal action a reconfiguration which will turn off some (combination of) not energy-efficient machines and replace them by turning on some (combination of) more energy-efficient machines, according to the considered future load. As those actions can consist in an important reconfiguration, we want to perform it only towards a quite stable situation, and only if the reconfiguration overhead is not too high. That is why we use long term look-ahead windows.

We define $window_{MAX}$ as the maximum look-ahead window used to decide energy-saving actions, and also used to check if the future load is globally increasing or decreasing. This window starts at $\min({t_{On}}^i)$ and ends at $\max({t_{On}}^i + T_s^i)$. For all architectures i we note $window_{long}^i$ the look-ahead window starting at ${t_{On}}^i$ and ending like $window_{MAX}$ at $\max({t_{On}}^i + T_s^i)$. There are all represented on Figure 6.3, and we explain in the following their use in the algorithm.

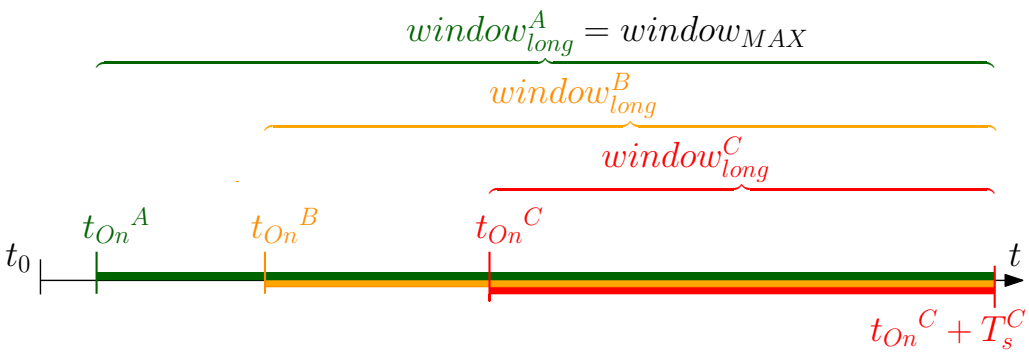


Figure 6.3: Long look-ahead windows used for Back-to-Ideal actions.

The decision of Back-to-Ideal action is only taken if no load-reactive actions have been proposed, and if there are currently no other ongoing reconfigurations, i.e., switch on/off actions not yet completed. Of course, another prerequisite of performing such an action is that the current combination of machines should not already be an ideal combination for any of the look-ahead windows.

Assuming these conditions are fulfilled, it exists two different situations when a Back-to-Ideal action is needed: (i) The load decreases in the future, and the current combination is sufficient to process it, but is also over-provisioned in a way that it is not possible to turn off any machines because there are all utilized. (ii) The load increases in the future, meaning that we will need to turn on new machines but the current combination is already far from ideal considering energy consumption.

Here is the detailed process to decide a Back-to-Ideal action:

1. Compute the load prediction for $window_{MAX}$, and the reconfiguration $reconfIdeal$ towards the associated ideal combination.

2. **If** the load prediction is lower than the processing capacity of the current infrastructure, the **load is decreasing**,
   Then:

   (a) Compare all consecutive $window^i_{long}$ to assure that the load is monotonically decreasing. (Else, exit).

   (b) Analyze $reconfIdeal$ to know which architecture $arch$ taking part in this reconfiguration has the maximum switch-on duration t^{arch}_{On}.

   (c) **If** t^{arch}_{On} is different from $\min({t_{On}}^i)$, it means that it is not optimal to perform the reconfiguration towards the ideal combination for $window_{MAX}$ but instead $window^{arch}_{long}$ should be considered;
   Then compute the load prediction on $window^{arch}_{long}$ and update $reconfIdeal$ to the new associated reconfiguration.
   (**Else**, keep the previous $reconfIdeal$).

   (d) Compute the energy overhead of the reconfiguration $reconfIdeal$: the sum of all Switch-On and Switch-Off energy overheads.

   (e) Compute the over-consumption due to staying in the current combination for the predicted load.

   (f) **If** the overhead of the reconfiguration is lower than the over-consumption of the current combination:
   Then perform Switch-On actions of $reconfIdeal$.
   (**Else**, exit).

3. **Else**, it means that the load prediction is greater than the processing capacity of the current infrastructure, hence the **load is increasing**,
   Then:

   (a) Check **if** the current combination of machines is already far from ideal; It consists in verifying that any quantity of machines of type i is greater than $maxNb^i$. (**Else**, exit).

   (b) Compare all consecutive $window^i_{long}$ to assure that the load is monotonically increasing. (Else, exit).

   (c) Same as Step 2) b).

   (d) Same as Step 2) c).

   (e) Perform Switch-On actions of $reconfIdeal$.

Note that we only perform the Switch-On actions of the chosen Back-to-Ideal because the switch-off actions will automatically be done later as a load-reactive action.

6.3 Comparative Evaluations of the two Algorithms

6.3.1 With Synthetic Traces

As we designed Multi-Terms algorithm to answer some limitations of the previously described Ideal BML algorithm, it is logical to evaluate this second algorithm by comparison with the first one. Note that QoS is not discussed in this chapter as we consider perfect knowledge of future load, and use look-ahead windows enough long to perform reconfigurations in time.

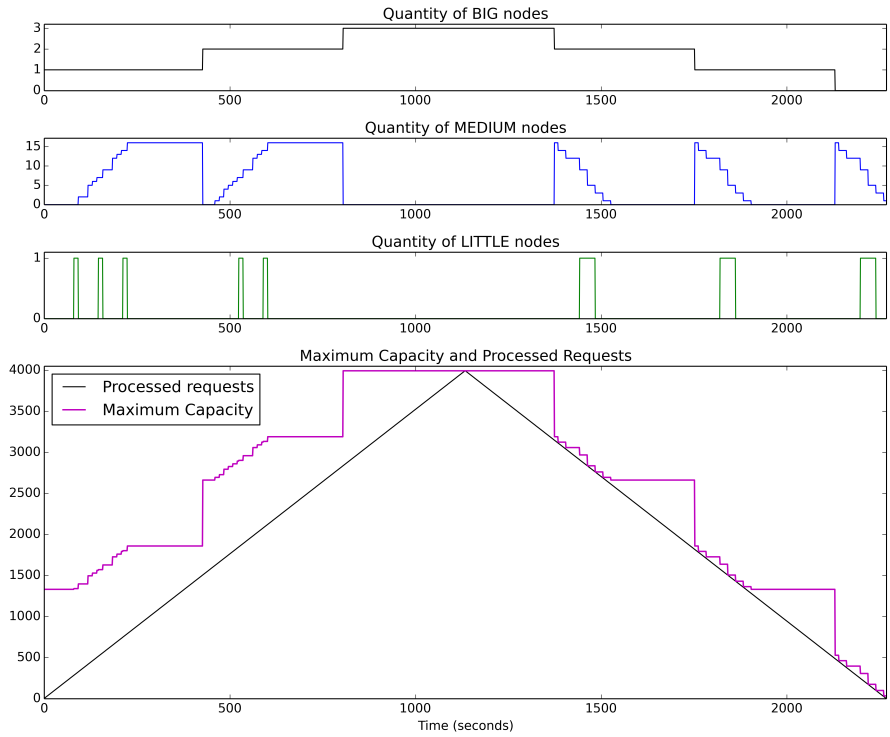
For a first comparison, we decide to use two simple generated traces consisting in two successive upward and downward phases of equal lengths. Both of them starts from a request rate of 0 to a maximum request rate of 3993 requests per second (corresponding to 3 times the maximum processing capacity of *Big* architecture), and down again to 0. We have generated two versions: a *low slope* with each phase lasting 2267 seconds, (which is the same we used in section 5.3), and a *steep slope* which has the same peak rate but is two times shorter as each phase lasts 1134 seconds.

Figure 6.4 presents the results for the *steep slopes*. In this case, Generic Multi-Terms algorithm gives better results than Ideal BML concerning total energy consumption and energy efficiency as the Joules per Request (JpR) metrics is 0,1797 against 0,1838. We can observe from visual comparison of the two graphs that Multi-Terms algorithm reduces the over-provisioning during the upward slope that was due to the long look-ahead window of Ideal BML algorithm. This comes from Load-Reactive actions that switch on *Little* nodes progressively as the load increases. Because the slope is steep, the system can predict the high raise and decide to switch on *Big* machines in advance.

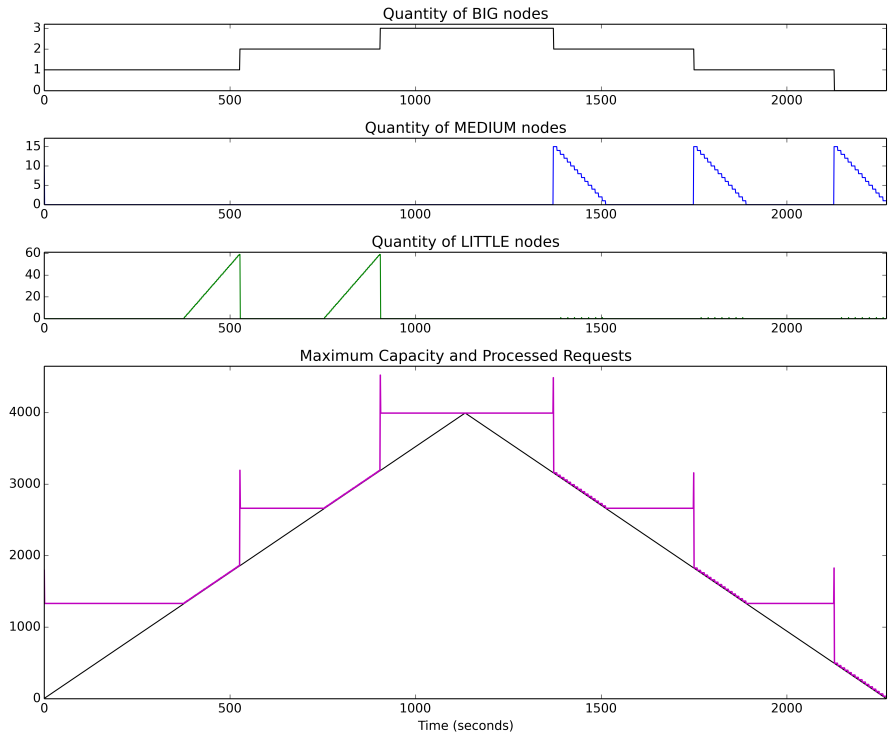
Figure 6.5 displays the results for the *low slopes*. We realize that the two algorithms perform very differently, and we conclude from the energy metrics that Multi-Terms version produces worse results than Ideal BML as JpR is 0,2503 against 0,1730. This significant difference can be easily explained by the quantity of *Little* nodes that explodes during the upward slope and reaches 396. The increasing slope is too low and the scheduler keeps switching on *Little* nodes to answer the little load raise. The important issue is that our algorithm prioritize Load-Reactive actions as Back-to-Ideal actions are only performed if no other actions need to be done. We observe that this characteristic of our Multi-Terms algorithm can be a disadvantage for this very specific case.

Of course this behavior is a consequence of our assumption on unlimited computing resources of each type. With a limited data center topology, this result would not be possible as the scheduler would not be able to switch on as many *Little* nodes and would consequently switch on available machines of the architectures sorted just after, in our case *Medium*, and then *Big*. This remark, as well as the comparative results of Multi-Terms algorithm against Ideal BML with real traces that we comment just after in 6.3.2, strengthens the many advantages of this scheduler version.

Regarding the downward slope, Multi-Term algorithm performs similarly no matter the gradient of the slope. Indeed, as the load decreases, Load-Reactive actions are decided, resulting in switching off unused machines. Once all of them have been switched off, if the load continues to decrease, then the scheduler will decide to perform a Back-to-Ideal action consisting in switching on smaller nodes. This will allow to switch off one of the biggest node, then be able to progressively switch off the smaller machines afterward.

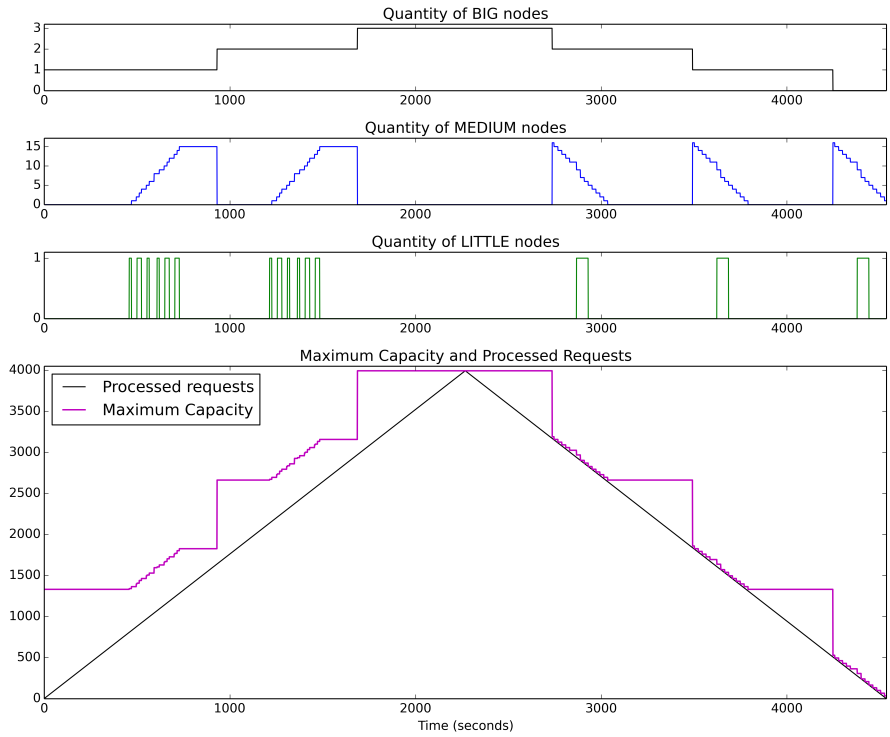


(a) **Ideal BML** - Total Energy Consumption: 832 692 J - JpR: 0,1838

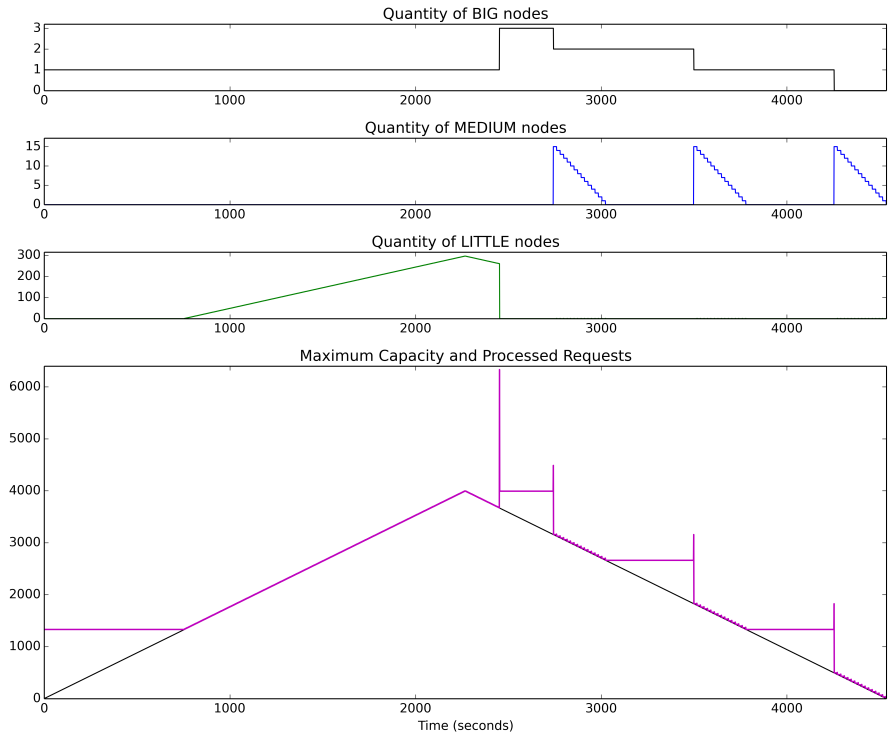


(b) **Multi-Terms BML** - Total Energy Consumption: 814 046 J - JpR: 0,1797

Figure 6.4: *Steep Slopes*: Multi-Terms performs *better* than Ideal BML.



(a) **Ideal BML** - Total Energy Consumption: 1 566 862 J - JpR: 0,1730



(b) **Multi-Terms BML** - Total Energy Cons.: 2 266 933 J - JpR: 0,2503

Figure 6.5: *Low Slopes*: Multi-Terms performs *worse* than Ideal BML.

6.3.2 With Real Traces

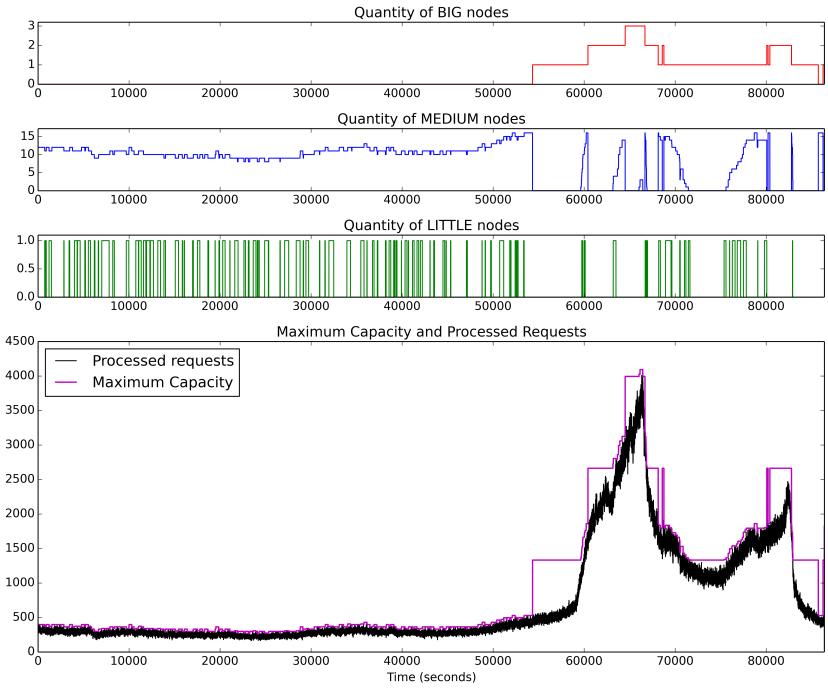
Figure 6.6 compares the results of the two algorithms for day 65 of the 1998 World Cup traces [1]. Day 65 has an average requests rate of 639 requests per second, while having a maximum peak rate of 4071 requests per second. We can observe on Figures 6.6(a) and 6.6(b) that the load profile of this day is quite low until approximately 50000 seconds, and contains two successive load peaks afterward. Table 6.6(c) gathers metrics to evaluate in details how the behaviors of the two algorithms differ.

As we already explained with the synthetic traces, the main difference between the two versions is the maximum quantity of utilized *Little* machines. For Ideal BML algorithm this quantity does not overpass 1 as it is the maximum number of *Little* nodes in pre-computed Ideal machine combinations. As for Multi-Terms algorithm, we consider that we can utilize as many machines as needed from each architecture types. In the case of day 65, it results in using until 41 *Little* machines. For *Big* and *Medium* machines, the maximum quantities are the same for both algorithms, respectively 3 and 16.

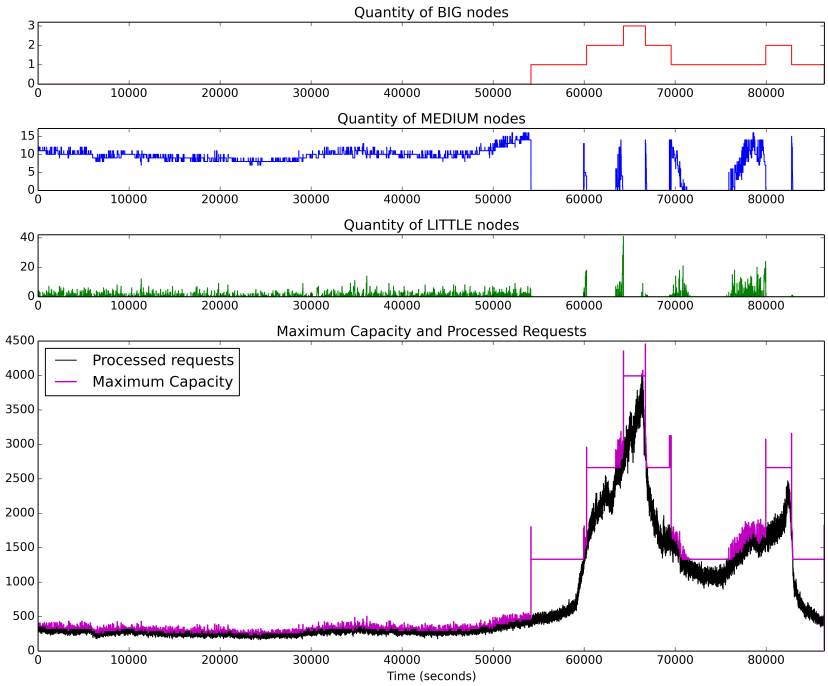
Multi-Terms algorithm gives a total energy consumption lower than Ideal BML algorithm. If we separate this total with compute energy on one side, and energy consumed during switch on/off actions on the other, we observe that the reduction is made on the compute part, while the on/off part being almost doubled. Of course this is due to the higher number of *Little* nodes utilized and the consequent higher number of reconfigurations actions. Multi-Terms algorithm reaches an infrastructure utilization over 81%, against 76% for Ideal BML version. Even if many *Little* machines are not considered *"ideal"*, using them permits to adapt the infrastructure's processing capacity very finely and increases its utilization while reducing the overall consumed energy for the whole day.

Figure 6.7 gathers comparative results concerning the two algorithms and the lower bound for all days of the World Cup traces. On average for the 86 days, Multi-Terms consumes -6% less than Ideal BML, the maximum difference and thus best result is -24.4% for day 22, and minimum difference or worst result is -0.5% for day 61. Table 6.7(a) also presents the difference percentage of Multi-Terms algorithm compared to the same theoretical lower bound we presented in section 5.5. The comparison of Ideal BML with upper bounds has already been done in this same section 5.5, and as the two algorithms provide quite close results, we focus this comparison only on them.

Figure 6.7(b) represents graphically the daily energy consumption differences between the two algorithms and this lower bound, while Figure 6.7(c) shows the comparison regarding daily energy proportionality. We can see that Multi-Terms BML stands just under Ideal BML as it increases the energy reduction by 6%. The maximum quantities of machines for Ideal BML are 3 *Big*, 16 *Medium* and 1 *Little* as we only authorize ideal combinations. Regarding Multi-Terms, the maximum numbers of utilized *Big* and *Medium* machines are the same as with Ideal BML, while the maximum number of *Little* nodes is 59.



(a) **Ideal BML**



(b) **Multi-Terms BML**

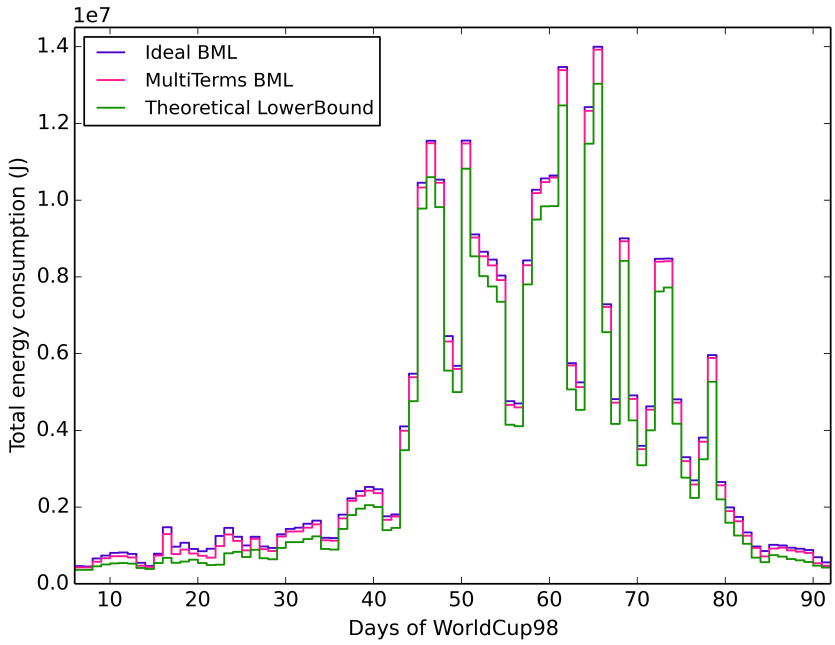
Algorithm	Total Energy	Compute Energy	On/Off Energy	JpR	%Util.	Nb Reconf	Max Nb [B,M,L]
Ideal BML	12430959.6	12240790.7	190168.9	0.1998	76.3%	294	[3,16,1]
Multi-Terms	12332962.2	11964347.6	368614.6	0.1982	81.1%	2809	[3,16,41]

(c) Evaluation Metrics

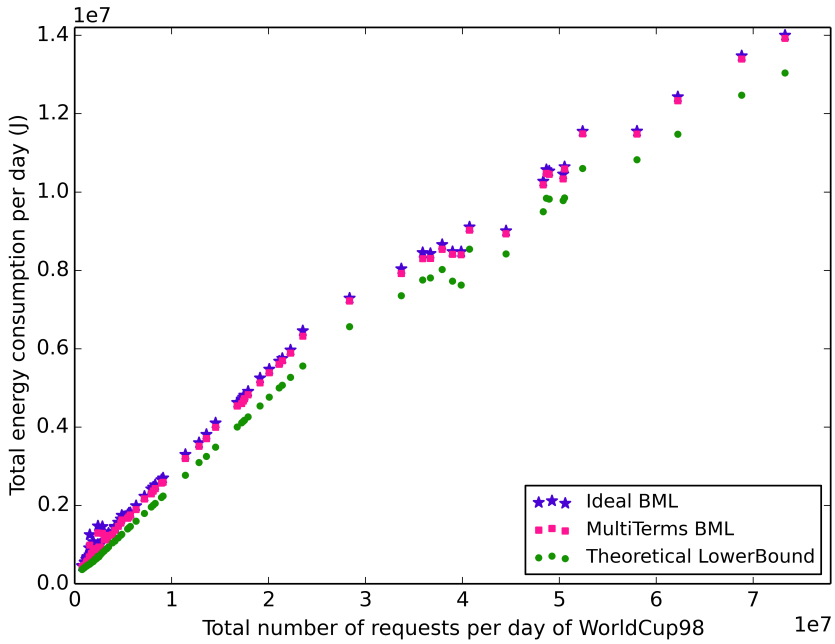
Figure 6.6: **Day 65** - Comparison between Ideal BML and Multi-Terms algorithms.

Energy Difference	Compared to Ideal BML	Compared to Lower Bound
Best *per day*	-25.2 % (day 22)	+5.6% (day 46)
Worst *per day*	-0.5 % (day 61)	+97.2 % (day 23)
Average *per day*	-6 %	+21.9 %

(a) **Total energy difference percentages**



(b) **Total energy consumption comparison**



(c) **Energy proportionality comparison**

Figure 6.7: **All days** - Comparison between Ideal BML and Multi-Terms algorithms and the lower bound.

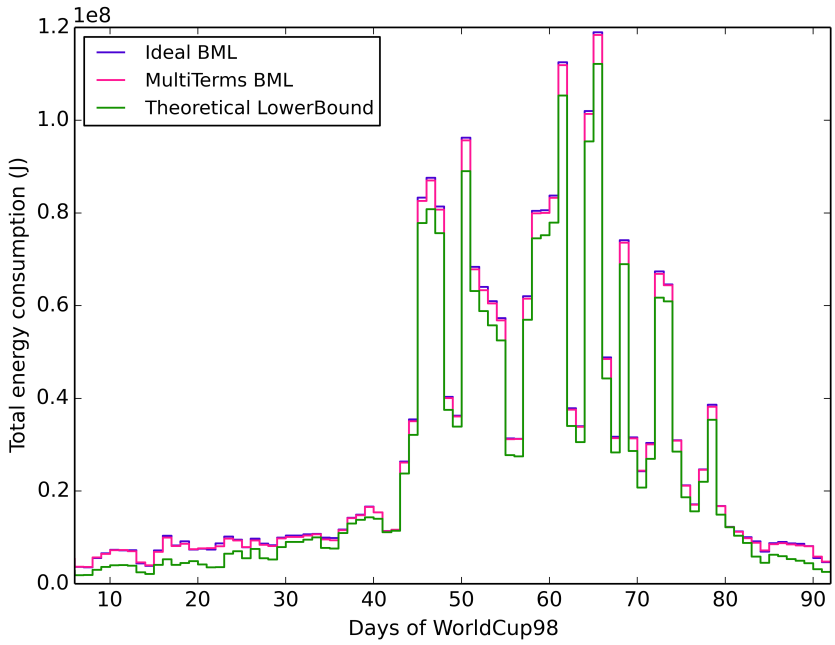
6.4 Results at Larger Scale

With the original traces of the 1998 World Cup, there are some days where no *Big* machines are needed but only combinations of *Medium* and *Little* nodes are sufficient to sustain the load. To provide evaluations at larger scale, we decided to create a modified version of these traces by multiplying all the original request rates by ten. We will refer to this modified traces as *Traces x10* in opposition to the *Original Traces*.

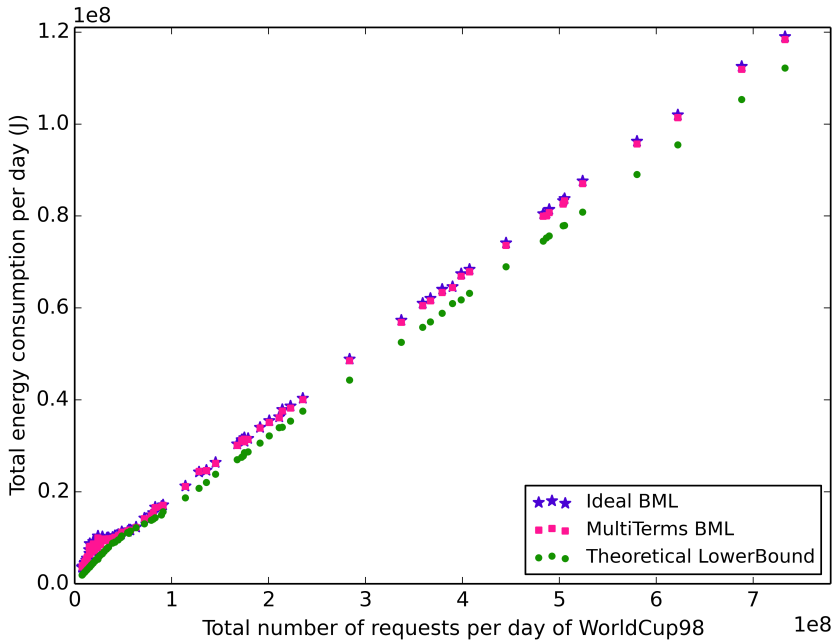
We have repeated the simulations with these large scale traces and Figure 6.8 provides the comparative results between the two algorithms and with the lower bound. As stated in Table 6.8(a), on average Multi-Terms consumes -0.85% less than Ideal BML, the best difference is -7% for day 23. For some days, Multi-Terms consume more than Ideal BML, the maximum positive difference, and worst result, is +6% for day 91. The maximum number of machines utilized for all days is 31 *Big*, 16 *Medium* and 1 *Little* for Ideal BML version, whereas Multi-Terms utilizes same number of *Big* and *Medium* nodes, but 130 *Little* machines.

Energy Difference	Compared to Ideal BML	Compared to Lower Bound
Best *per day*	-7 % (day 23)	+0.64% (day 81)
Worst *per day*	+6 % (day 91)	+123.8 % (day 23)
Average *per day*	-0.85 %	+33.5 %

(a) **Total energy difference percentages**



(b) **Total energy consumption comparison**



(c) **Energy proportionality comparison**

Figure 6.8: **All days - Traces x10** - Comparison between Ideal BML and Multi-Terms algorithms.

6.5 Big-Medium versus BML

As we have done in previous chapter in section 5.4, we want to evaluate the gains brought by heterogeneity with comparative evaluations of our algorithm with the full BML infrastructure against the same algorithm but with only *Big* and *Medium* machines.

Unfortunately, the doubts we had concerning the interest of the *Little* architecture were confirmed. Multi-Terms algorithm and BML infrastructure results in more energy consumption than this algorithm with an infrastructure composed of *Big* and *Medium* machines. On average on all original days of the World Cup traces, Multi-Terms BML consumes 2.4% more than Multi-Terms Big-Medium, the best result being -2.5% for day 85 and the worst +27.2% for day 23.

These disappointing results come from the fact that *Little* infrastructure has a switch on duration longer than *Medium*'s. The *Little* machine (*Raspberry Pi*) takes 16 seconds to power on, whereas the *Medium* architecture (*Chromebook*) takes 12 seconds. Indeed, the sorting and tagging done during the building of BML combinations only consider performance and power, but not the switch on durations. The minimum utilization threshold of *Little* machine is set to 1 and thus for any load increase, *Little* machines will be switched on. The minimum utilization of *Medium* is set accordingly to its performance and power ratio compared to *Little*, which is 10 requests per second in our case. Consequently, for load increases greater than 10 requests per second, *Medium* machines will be powered on in addition of *Little* ones. This issue is that these *Little* nodes will in fact be ready to process requests after *Medium* nodes, and will then be useless. This explains how in this specific infrastructure, more heterogeneity does not bring better results. These experiments allow us to highlight another important characteristic required for building a relevant BML infrastructure which is that switch on durations should follow the same order as performance and power.

To demonstrate this affirmation, we have modified the profile of the *Little* architecture by divided by two its switch on and off durations and energy consumption. This translates in two BML infrastructures defined as follows:

- *Real BML*: Existing hardware profiles as detailed in Tables 4.2 and 4.3.
- *Modified Little BML*: Existing hardware profiles for *Big* and *Medium* architectures, and *Little*'s on and off durations and energy overheads divided by two.
  The new values are:
  t_{On}^{Little}: 8 s, e_{On}^{Little}: 20.25 J; t_{Off}^{Little}: 7 s, e_{Off}^{Little}: 18.1 J; and T_s: 15 s.

We have run the simulations with original traces for Multi-Terms algorithm with the *Modified Little BML* infrastructure, and the results compared to Big-Medium infrastructure are much more encouraging. On average Multi-Terms with *Modified Little BML* consumes -1.6% less than Multi-Terms Big-Medium. The best result is -8.1% for day 85 and the worst is +0.3% for day 48. Table 6.1 gathers the comparative results of the two different BML infrastructures against Big-Medium scenario. Even with *Modified Little BML* there are some days where the introduction of *Little* architecture implies a small increase in energy consumption compared to Big-Medium. This is still due to the small range of utilization of the *Little* architecture as we already discussed in section 5.4.

Moreover, another difference with the *Modified Little BML* scenario compared to *Real BML* is the utilization of the different architectures types. The maximum number of *Big* machine is still 3, but 27 *Medium* nodes and 6 *Little* are used (compared to respectively 16 and 59 for *Real BML*).

Table 6.1: Total energy difference of Multi-Terms algorithm version BML Normal and version BML with Modified Little compared to Multi-Terms algorithm with Big-Medium infrastructure

Energy Diff. to BM	BML Real	BML with Little Modified
Best *per day*	-2.5 % (day 85)	-8.1% (day 85)
Worst *per day*	+27.2 % (day 23)	+0.3 % (day 48)
Average *per day*	+2.4 %	-1.6 %

– Chapter 7 –

Discussions

This chapter aims at answering some questions the reader may have at this point, as well as discussing some topics around our approach that we consider important. The goal is to place our work in a larger perspective by addressing issues such as implementation practicability, potential target applications and use cases, evolution of our proposition with future hardware, and impacts on the data center considered as a whole.

7.1 Are Ideal Combinations really Ideal?

The first algorithm we designed, Ideal BML, lays on the ideal combinations computed based on hardware profiles. We have decided to build those combinations only with performance and power consumption information. Consequently, the main characteristic of these combinations is that they are ideal if we consider them as instantaneous configurations for given performance rates. Depending on the current combination of powered on machines, the ideal combination for the predicted load may not necessarily be one of the pre-computed ideal combinations. Indeed, it depends on the longevity of this predicted performance rate – is it just a brief peak/drop? or will the load continue increasing/decreasing after? – and also on the energy overheads of the switch on and off actions needed to reconfigure the infrastructure towards the ideal combination.

This observation has lead us towards designing the second algorithm, Multi-Terms, that is not limited by pre-computed ideal combinations and takes into account the different temporalities as well as the overheads of the different reconfiguration actions. However, with our assumption of unlimited quantity of machines of each architecture type, we have shown that for some specific load patterns, it leads to a high number of utilized *Little* machines and thus a higher energy consumption. Of course, this behavior would not be possible in practice with a fixed heterogeneous data center topology, but this raises the following question: if we authorize more machines than the ones composing pre-computed ideal combinations, what is the optimal quantity of machines of each type the data center should contain? It is not easy to compute as it is dependent on the different characteristics of the hardware profiles and on the load evolutions.

7.2 What are the Impacts of Load Prediction Errors?

In our work we consider having a perfect knowledge of the application workload over time. In practice, this eventuality is very rare, and in most cases, future workload would be predicted thanks to models based on past workload data. Prediction models represent a whole research field and is out of the scope of this thesis. Nevertheless, it is necessary to discuss about the potential impacts of prediction errors on our approach. Depending on many parameters – the application, the knowledge of the user demand, the influence of unpredictable events - prediction models can be more or less accurate.

A classical approach to prevent QoS degradation due to prediction errors is over-provisioning. If the infrastructure is dimensioned for a greater capacity than the predicted load, in case of underestimation it would be ready to process it, but in case of overestimation the infrastructure would be even more over-provisioned, resulting in energy wasting that we want to avoid. Regarding our work, Multi-Terms algorithm is more suitable to cope with prediction errors as it makes reconfiguration actions which take effect at different moments in the future. Indeed, this algorithm can provision the infrastructure for long-term load predictions with machines having longer switch on durations, and then using machines with fast switch on actions to adjust the infrastructure's capacity to more accurate short-term predictions that may differ from the previous long-term vision. With the unique look-ahead window of Ideal BML algorithm, over-provisioning on the long-term would be the only solution to respect QoS requirements with prediction errors, and would consequently consumes more energy than Multi-Terms algorithm.

7.3 What can be the other Target Applications Types?

In this thesis, we focus on a specific application that is a stateless web server. As it relaxes many constraints, it allows to demonstrate the full benefits of our heterogeneous BML infrastructure. The question of applicability of this approach to other types of application is predominant. THul applications raise the issue of longer and more expensive migrations due to the amount of data to transfer. Moreover, the size of the state can vary during the life of the application, thus the duration of migration too, and this complicates how to take these extra overheads in consideration for reconfigurations decisions. Another parameter which can influence the migration costs is the number of machines in the current combination versus the future combination, as the number of application instances depends on the number of available nodes.

Furthermore, not all applications are malleable and the number of instances of an application can be fixed. In such cases, BML combinations and the general way of provisioning the infrastructure will be constrained by this number. The resulting energy savings will be a little reduced compared to those for stateless web servers as the number of possible combinations of machines will be bounded.

We can also imagine dealing with multi-tiers applications. Each tier can be profiled on each type of hardware and have different scheduling schemes depending on their specific load evolutions. In any case, the essential characteristics for target applications are to have a variable workload over time, and the ability to be executed and migrated across different architectures.

7.4 How will BML evolve with Hardware Evolutions?

The implementation of BML infrastructure that we have presented in this thesis has been built with pieces of hardware which we have chosen and which were available at that time. Since then, new hardware has appeared with even better energy efficiency characteristics. Some examples are *ARM 64-bits* processors, available in the last generation of *Raspberry Pi 3*, or the *Intel Core M* series of processor engraved with the 14nm (nanometer) technology. It would be interesting to profile them and see how much energy gains they can bring in a heterogeneous infrastructure.

Other kinds of hardware offer attractive energy efficiency, like *GPU*, *FPGA (Field-Programmable Gate Array)*, or *DSP (Digital Signal Processor)*. However, they are very specialized and are not suitable or relevant for all types of workload. Moreover, most of them require dedicated programming languages and thus the target application needs to have a version of its program written specifically for one of this processor to be able to include it in the infrastructure.

The technological improvements continue and more and more powerful and energy efficient hardware will be built in the future. Even if we do not know if a perfectly energy proportional server will be developed one day, one can wonder if our work will still be profitable with very efficient and almost proportional hardware. Our point of view is that there are good chances that heterogeneity will still exist in the future. With different application types having different performance affinities and energy consumption with each type of processors, heterogeneity aware scheduling is always needed. Also at a very large scale, even if idle power consumption is low, switching off unused machines can lead to important energy savings, and different switch on durations need to be known and considered to turn them back on in time efficiently.

7.5 Can the Whole Data Center be Energy Proportional and Sustainable?

In our work, we focus only on the energy proportionality of the computing resources. A similar approach can be applied to the other high consuming component of the data center that is the cooling system.

With an energy proportional infrastructure, meaning that its energy consumption follows the evolutions of the workload, it also implies that the heat generated by the servers vary over time, and consequently the air conditioning system needs to be configurable to adjust its cooling power. Indeed, too much heat can be prejudicial for IT equipments, but too much cold also. That is why during periods of low load when only low power machines are powered on, the cooling system can not continue to cool the data center room as if all servers were on.

It is necessary to develop a coordinate approach between computing resources and the cooling system in order to reach energy proportionality for the whole data center. Without neglecting that the energy provider and power supply installations should be aware and adapted to variations.

Building a BML infrastructure implies that we are adding more machines in the data center that are not always utilized. The relevance of this solution can be questioned regarding sustainability: Are the energy savings made when powering computing resources greater than the additional cost of producing, shipping and installing these machines in the data center?

Life Cycle Analysis (LCA) is a procedure that assess the environmental repercussion of a system by quantifying the impacts of all its life's stages from raw material extraction, manufacturing, distribution, use, maintenance, to disposal or recycling. In a first place, it would be very interesting to be able to know the full *LCA* of a data center. And in a second place, it would be valuable for our work to evaluate the *LCA* of our BML infrastructure against a typical homogeneous data center.

– Chapter 8 –

Conclusions and Perspectives

This chapter summarizes the main contributions of this thesis and details the different perspectives that can be explored to pursue this work.

8.1 Conclusions

Because our world is relying more and more on internet services, companies are building more and more data centers. But these infrastructures have non negligible impacts on our environment, therefore it is necessary to reduce their energy consumption as much as possible to protect our planet.

The goal of this thesis was to find a way to achieve a data center whose energy consumption is proportional to its actual load. We wanted to tackle the issue of important static costs in classical data centers which are due to resource over-provisioning and servers' high idle consumption. We were inspired by the work done in the mobile world as hardware designers and software developers have to make efforts to limit the energy consumption of these devices which are battery-powered. We have decided to generalize the concept of the heterogeneous mobile processor *ARM big.LITTLE* to a larger scale by creating a highly heterogeneous data center, composed of different types of machines, from very low power mobile processors to very powerful servers. We name our approach *BML* for *"Big, Medium, Little"* to picture the generalization. The idea is to be able to dynamically adjust the composition of the infrastructure, and consequently its energy consumption, to the evolutions of the load over time.

Our contributions can be divided into two main parts:

- **Design of an Energy Proportional Heterogeneous Infrastructure**

  We have proposed a step by step approach for building an energy proportional infrastructure with heterogeneous computing resources which are chosen for their energy efficiency. It begins by a profiling step whose objective is to measure the performance and energy consumption characteristics of the different types of available hardware for the target application. Then the following steps analyze the different profiles to find how the machines should be combined to achieve the most

energy proportional infrastructure. The final step computes what we call the ideal machines combinations. The methodology we have defined is generic and can be adapted to any number of architectures.

- **Dynamic Provisioning of Heterogeneous Computing Resources**

  Once the composition of the infrastructure has been defined, an efficient scheduling and provisioning algorithm is needed to take advantage of the different characteristics of the heterogeneous computing resources. We have developed two versions of provisioning algorithm: the first one, *Ideal BML* that only considers pre-computed machines combinations as configuration possibilities; and the second one, *Multi-Terms*, that performs more complex reconfigurations as it takes into account the different temporalities and overheads of the different switch on and off actions in the decision process.

We have performed profiling experiments on a set of real hardware, for the chosen use case application which is a stateless web server. Our methodology has allowed us to keep only the most energy efficient ones, and lead to an implementation of the BML infrastructure with three different types of architectures. To evaluate our provisioning algorithms, we have developed a simulator based on the machines profiles acquired experimentally. It enables to try multiple infrastructure configurations as well as different algorithm settings, for any trace file given as input.

Consequently, we have been able to demonstrate the energy gains of our solution against classical over-provisioned data center composed of homogeneous servers, for real workload traces of a web site. We have shown that BML infrastructure with our dynamic provisioning algorithms manage to save significant amounts of energy, especially during periods of low loads when classical data centers suffer from high static costs. Our solution reaches energy proportionality as we have proved that its results are very close to the theoretical lower bound corresponding to a perfect provisioning of BML resources without considering any energy and time overheads from the reconfigurations.

Finally, we have provided a comparative study of the two provisioning algorithms we proposed, and discussed the differences between the two approaches which mostly concern the consideration of ideal machines combinations. We have demonstrated that Multi-Terms algorithm achieves the best results as it allows to adjust more finely the infrastructure to the load. We also assume that this characteristic would make it more robust to load prediction errors, however more evaluations are needed to prove it experimentally. This is considered as future work, as well as the other perspectives detailed in the following section.

8.2 Perspectives

Our contributions can be extended in several directions. Many of them concern the different topics to address to be able to actually implement our approach.

The first one would be to deal with a fixed data center topology. Only small modifications in our BML building methodology and dynamic provisioning algorithm are needed to consider an already built heterogeneous infrastructure. In such cases, we would not remove from consideration the least energy efficient hardware, but we would keep them

and sort them in a list of second choice machines which would be switched on only when no more efficient machines are available. It would allow to take the most advantage from the infrastructure when possible, and limiting the static costs due to powering the least energy efficient machines.

Another step toward actual implementation would be to face workload which is not known in advance. This would require using a load prediction system fitted to the target application, and adapting the provisioning algorithm to cope with the eventual prediction errors. We can imagine adding a little over-provisioning to our approach. It can consist in considering a greater load of a certain percentage of the prediction, or even leaving some *Little* machines always on if they have a very small idle consumption. Experiments and evaluations are needed to determine which solution would be the most energy proportional while offering a good quality of service for the application.

Extending our approach to other types of applications is another direction that we want to address. We worked with stateless web servers in this thesis because they relax some constraints that were difficult to consider in a first step, but we know this kind of application is quite rare in reality. Stateful applications are the next type of target applications that we want to consider. They add the novelty of a state to carry with the application while migrating it, but they can be fully malleable like the stateless applications we consider until then. The size of the state will have an important impact on the duration of the migration. We need to add a profiling phase considering the network in order to predict how long will last the migration depending on the size of the state and the two, or more, machines impacted by the transfer. If the size of the state varies over time, its evolution would be another parameter that the prediction model would need to anticipate.

Publications

International Journals

[J1] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. “"Big, Medium, Little": Reaching Energy Proportionality with Heterogeneous Computing Scheduler”. In: *Parallel Processing Letters* 25.3 (2015).

International Conferences

[C1] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. “Dynamically Building Energy Proportional Data Centers with Heterogeneous Computing Resources (short paper)”. In: *IEEE International Conference on Cluster Computing (Cluster 2016), Taipei, Taiwan, September 13-15*. 2016, pp. 217–220.

[C2] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. “Energy Aware Dynamic Provisioning for Heterogeneous Data Centers”. In: *IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2016), Los Angeles, USA, October 26-28*. 2016.

[C3] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. “Energy Proportionality in Heterogeneous Data Center Supporting Applications with Variable Load”. In: *IEEE International Conference on Parallel and Distributed Systems (ICPADS 2016), Wuhan, China, December 13-16*. 2016.

[C4] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. “Towards Generalizing "Big Little" for Energy Proportional HPC and Cloud Infrastructures”. In: *IEEE Fourth International Conference on Big Data and Cloud Computing (BDCloud 2014), Sydney, Australia, December 3-5*. 2014, pp. 703–710.

National Conferences

[NC1] V. Villebonnet, G. Da Costa, L. Lefèvre, J-M. Pierson, and P. Stolf. *Généralisation du concept "big.LITTLE" pour aller vers des infrastructures cloud énergétiquement proportionnelles*. Conférence d’Informatique en Parallelisme Architecture et Système (ComPas 2015), Lille, France, Juillet 1-3, 2015.

Bibliography

[1] 1998 World Cup Web Site Access Logs. *http://ita.ee.lbl.gov/html/contrib/WorldCup.html*.

[2] Advanced Configuration and Power Interface(ACPI) Specification Version 6.1. *http://www.uefi.org/*. 2016.

[3] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H.S. Lee. "Idlepower: Application-Aware Management of Processor Idle States". In: *In Proceedings of MMCS, in conjunction with HPDC'08*. 2008.

[4] D. Ardagna, S. Casolari, and B. Panicucci. "Flexible Distributed Capacity Allocation and Load Redirect Algorithms for Cloud Systems". In: *IEEE Int. Conf. on Cloud Computing*. 2011.

[5] M. Arlitt and T. Jin. *Workload Characterization of the 1998 World Cup Web Site*. Tech. rep. IEEE Network, 1999.

[6] *ARM big.LITTLE: The Future of Mobile*. 2013.

[7] D. Balouek-Thomert, E. Caron, and L. Lefèvre. "Energy-Aware Server Provisioning by Introducing Middleware-Level Dynamic Green Scheduling". In: *HPPAC'2015*. Hyderabad, India, May 2015.

[8] L.A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, 2013.

[9] L.A. Barroso and U. Holzle. "The Case for Energy-Proportional Computing". In: *IEEE Computer* (2007).

[10] R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez, E. Jeannot, Y. Jégou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quétier, O. Richard, T. El-Ghazali, and I. Touche. "Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed". In: *International Journal of High Performance Computing Applications* 20.4 (2006).

[11] G.L. Tsafack Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf, and G. Da Costa. "Application-Agnostic Framework for Improving the Energy Efficiency of Multiple HPC Subsystems". In: *23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2015, Turku, Finland, March 4-6, 2015*. 2015, pp. 62–69.

[12] Christmann. *Description for Resource Efficient Computing System (RECS)* `http://shared.christmann.info/download/projectrecs.pdf`. 2009.

[13] B-G. Chun, G. Iannaccone, G. Iannaccone, R. Katz, G. Lee, and L. Niccolini. "An Energy Case for Hybrid Datacenters". In: *SIGOPS Oper. Syst. Rev.* 44.1 (Mar. 2010), pp. 76–80.

[14] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. "Live Migration of Virtual Machines". In: *Conference on Symposium on Networked Systems Design & Implementation.* 2005.

[15] J. Cong and B. Yuan. "Energy-Efficient Scheduling on Heterogeneous Multi-Core Architectures". In: *Proceedings of ACM/IEEE International Symposium on Low Power Electronics and Design.* ISLPED '12. ACM, 2012, pp. 345–350.

[16] G. Da Costa. "Heterogeneity: The Key to Achieve Power-Proportional Computing". In: *IEEE International Symposium on CCGrid* (2013).

[17] Dall, C. and Nieh, J. "KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor". In: *SIGARCH Comput. Archit. News* 42.1 (2014), pp. 333–348.

[18] W. Dargie. "Analysis of the Power Consumption of a Multimedia Server under Different DVFS Policies". In: *IEEE International Conference on Cloud Computing (CLOUD).* 2012, pp. 779–785.

[19] K. Ebrahimi, G.F. Jones, and A.S. Fleischer. "A Review of Data Center Cooling Technology, Operating Conditions and the Corresponding Low-Grade Waste Heat Recovery Opportunities". In: *Renewable and Sustainable Energy Reviews* 31 (2014), pp. 622 –638.

[20] N. El-Sayed, I.A. Stefanovici, G. Amvrosiadis, A.A. Hwang, and B. Schroeder. "Temperature Management in Data Centers: Why Some (Might) Like It Hot". In: *SIGMETRICS Perform. Eval. Rev.* 40.1 (2012), pp. 163–174.

[21] S. Rivoire *et al.* "A Comparison of High-level Full-system Power Models". In: *Conference on Power Aware Computing and Systems (HotPower)* (2008).

[22] J. Enos, C. Steffen, J. Fullop, M. Showerman, G. Shi, K. Esler, V. Kindratenko, J.E. Stone, and J.C. Phillips. "Quantifying the Impact of GPUs on Performance and Energy Efficiency in HPC Clusters". In: *International Green Computing Conference.* 2010, pp. 317–324.

[23] F. Fakhar, B. Javed, R. Rasool, O. Malik, and K. Zulfiqar. "Software Level Green Computing for Large Scale Systems". In: *Journal of Cloud Computing: Advances, Systems and Applications* 1.1 (2012), pp. 1–17.

[24] E. Feller, L. Rilling, and C. Morin. "Energy-Aware Ant Colony Based Workload Placement in Clouds". In: *IEEE/ACM International Conference on Grid Computing.* 2011.

[25] Google. *Efficiency: How we do it -* `https://www.google.com/about/datacenters/efficiency/internal/`. Q2 2016.

[26] L.D. Gray, A. Kumar, and H.H. Li. "Workload Characterization of the SPECpower_ssj2008 Benchmark". In: *Performance Evaluation: Metrics, Models and Benchmarks: SPEC International Performance Evaluation Workshop, Darmstadt, Germany, June 27-28, 2008. Proceedings.* Springer Berlin Heidelberg, 2008.

[27] The Green Grid. *Green Grid Metrics: Describing Datacenter Power Efficiency.* 2007.

[28] T. Heath, B. Diniz, E. V. Carrera, W. Meira Jr., and R. Bianchini. "Energy Conservation in Heterogeneous Server Clusters". In: *Proceedings of the Tenth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming.* PPoPP '05. 2005, pp. 186–195.

[29] F. Hermenier, X. Lorca, J-M. Menaud, G. Muller, and J. Lawall. "Entropy: A Consolidation Manager for Clusters". In: *ACM International Conference on Virtual Execution Environments.* 2009.

[30] C.-H. Hsu and S. W. Poole. "Measuring Server Energy Proportionality". In: *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering.* ICPE '15. Austin, Texas, USA: ACM, 2015, pp. 235–240.

[31] J. Kim and D. Rotem. "FREP: Energy proportionality for disk storage using replication". In: *Journal of Parallel and Distributed Computing* 72.8 (2012), pp. 960 –974.

[32] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz. "NapSAC: Design and Implementation of a Power-proportional Web Cluster". In: *SIGCOMM Comput. Commun. Rev.* 41.1 (2011), pp. 102–108.

[33] L. Cupertino and G. Da Costa and A. Oleksiak and W. Piatek and J-M. Pierson and J. Salom and L. Sisó and P. Stolf and H. Sun and T. Zilio. "Energy-Efficient, Thermal-Aware Modeling and Simulation of Data Centers: The CoolEmAll Approach and Evaluation Results". In: *Ad Hoc Networks* 25, Part B (2015), pp. 535 –553.

[34] Lighttpd Web Server. *http://www.lighttpd.net/*.

[35] D. Lo, L. Cheng, R. Govindaraju, L.A. Barroso, and C. Kozyrakis. "Towards Energy Proportionality for Large-scale Latency-critical Workloads". In: *SIGARCH Comput. Archit. News* 42.3 (2014).

[36] D. Loghin, B.M. Tudor, H. Zhang, B.C. Ooi, and Y.M. Teo. "A Performance Study of Big Data on Small Nodes". In: *VLDB* (2015).

[37] T. Lu, M. Chen, and L.L.H. Andrew. "Simple and Effective Dynamic Provisioning for Power-Proportional Data Centers". In: *IEEE Transactions on Parallel and Distributed Systems* (2013).

[38] K. Ma, X. Li, W. Chen, C. Zhang, and X. Wang. "GreenGPU: A Holistic Approach to Energy Efficiency in GPU-CPU Heterogeneous Architectures". In: *41st International Conference on Parallel Processing.* 2012, pp. 48–57.

[39] K.T. Malladi, I. Shaeffer, L. Gopalakrishnan, D. Lo, B.C. Lee, and M. Horowitz. "Rethinking DRAM Power Modes for Energy Proportionality". In: *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture*. Vancouver, B.C., Canada: IEEE Computer Society, 2012, pp. 131–142.

[40] U.F. Mayer. *Linux/Unix nbench*.

[41] D. Meisner, B.T. Gold, and T.F. Wenisch. "PowerNap: Eliminating Server Idle Power". In: *SIGARCH Computer Architecture News* 37.1 (2009).

[42] G. Mounie, C. Rapine, and D. Trystram. "Efficient Approximation Algorithms for Scheduling Malleable Tasks". In: *ACM symposium on Parallel algorithms and architectures*. 1999, pp. 23–32.

[43] R. Nathuji, C. Isci, and E. Gorbatov. "Exploiting Platform Heterogeneity for Power Efficient Data Centers". In: *International Conference on Autonomic Computing (ICAC'07)*. 2007.

[44] A-C. Orgerie, L. Lefevre, and J-P. Gelas. "Chasing Gaps between Bursts: Towards Energy Efficient Large Scale Experimental Grids". In: *Int. Conf. on Parallel & Distributed Comp., Applications & Technologies*. 2008.

[45] Hewlett Packard. *HP Moonshot System*. 2014.

[46] N. Penneman, D. Kudinskas, A. Rawsthorne, B. De Sutter, and K. De Bosschere. "Formal Virtualization Requirements for the ARM Architecture". In: *Journal of Systems Architecture* 59.3 (2013), pp. 144 –154.

[47] E. Pinheiro, R. Bianchini, E.V. Carrera, and T. Heath. "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems". In: *Workshop on compilers and operating systems for low power* (2001).

[48] N. Rajovic, P. M. Carpenter, I. Gelado, N. Puzovic, A. Ramirez, and M. Valero. "Supercomputing with Commodity CPUs: Are Mobile SoCs Ready for HPC?" In: *2013 SC - International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2013, pp. 1–12.

[49] F. Rossigneux, L. Lefevre, J-P. Gelas, and M. Dias de Assuncao. "A Generic and Extensible Framework for Monitoring Energy Consumption of OpenStack Clouds". In: *IEEE SustainCom*. 2014.

[50] F. Ryckbosch, S. Polfliet, and L. Eeckhout. "Trends in Server Energy Proportionality". In: *Computer* (2011).

[51] M. Sabharwal, A. Agrawal, and G. Metri. "Enabling Green IT through Energy-Aware Software". In: *IT Professional* 15.1 (2013), pp. 19–27.

[52] E. Saxe. "Power-Efficient Software". In: *ACM Queue* 8.1 (2010), 10:10–10:17.

[53] A. Shehabi, S.J. Smith, D.A. Sartor, R.E. Brown, M. Herrlin, J.G. Koomey, E.R. Masanet, N. Horner, I.L. Azevedo, and W. Lintner. *United States Data Center Energy Usage Report*. 2016.

[54] Siege Benchmark. `https://www.joedog.org/siege-home/`.

[55] B. Subramaniam and W-C. Feng. "On the Energy Proportionality of Distributed NoSQL Data Stores". In: *International Workshop in High Performance Computing Systems PMBS*. 2014.

[56] The Mont-Blanc Project. *https://www.montblanc-project.eu/*.

[57] N. Tolia, Z. Wang, M. Marwah, C. Bash, P. Ranganathan, and X. Zhu. "Delivering Energy Proportionality with Non Energy-proportional Systems: Optimizing the Ensemble". In: *Conference on Power Aware Computing and Systems (HotPower'08)*. 2008.

[58] Uptime Institute. *https://journal.uptimeinstitute.com/2014-data-center-industry-survey/*. 2014.

[59] B. Varghese, N. Carlsson, G. Jourjon, A. Mahanti, and P. Shenoy. "Greening Web Servers: A Case for Ultra Low-Power Web Servers". In: *International Green Computing Conference (IGCC)*. 2014.

[60] G. Varsamopoulos, Z. Abbasi, and S.K.S. Gupta. "Trends and Effects of Energy Proportionality on Server Provisioning in Data Centers". In: *International Conference on High Performance Computing*. 2010.

[61] D. Wong and M. Annavaram. "Implications of High Energy Proportional Servers on Cluster-Wide Energy Proportionality". In: *IEEE International Symposium on High Performance Computer Architecture, HPCA*. 2014.

[62] D. Wong and M. Annavaram. "Knightshift: Scaling the Energy Proportionality Wall Through Server-Level Heterogeneity". In: *IEEE/ACM MICRO* (2012).