Energy Proportionality in Heterogeneous Data Center Supporting Applications with Variable Load

Violaine Villebonnet*†, Georges Da Costa*, Laurent Lefevre†, Jean-Marc Pierson* and Patricia Stolf*
*IRIT, University of Toulouse, France
†Inria Avalon LIP - Ecole Normale Superieure of Lyon, University of Lyon, France

***Abstract*—The increasing number of data centers raises serious concerns regarding their energy consumption. Although servers have become more energy-efficient over time, their idle consumption remains high, which is an issue as resources in data centers are often over-provisioned. This work proposes a novel approach for building data centers so that their energy consumption is proportional to load. A data center hence comprises heterogeneous machines carefully chosen for their performance and energy efficiency ratios. We focus on web applications whose load varies over time and design a scheduler that dynamically reconfigures the infrastructure to minimize its energy consumption according to current load and application requirements. Based on load forecasts, it takes reconfiguration decisions and performs actions such as migrating applications and switching machines on or off. The approach is evaluated considering a data center with heterogeneous resources, and the experiments show how to adjust the parameters of scheduling policies to save the most energy while satisfying Quality of Service (QoS) constraints.**

***Index Terms*—Energy Proportionality; Heterogeneous Infrastructure; Dynamic Provisioning; Variable Load Applications;**

I. Introduction

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 infrastructures, and especially data centers, are responsible for a substantial amount of the greenhouse emissions. Data centers are often over-provisioned and contain numerous servers that are not fully utilized. An Uptime Institute survey [1] suggests that close to 30% of servers in US enterprises' data centers are *comatose*, meaning they are consuming power but not doing any useful work. When it is idle, a typical server can consume up to 50% of the power drawn at peak utilization [2]. There is therefore a need for novel approaches for conceiving and managing data centers more efficiently.

This work provides an approach for designing a data center whose energy consumption is proportional to its load. It focuses on services with variable load, and offers means for adjusting the computational capacity to service requirements. This adjustment consists in changing the infrastructure performance by modifying the number and type of machines hosting a service, and hence reducing the energy consumption and cost. The goal is to minimize the energy consumed by allocating the minimum number of resources required to meet the application demands.

We advocate the use of infrastructure comprising multiple server architectures that feature heterogeneous ranges of performance and energy consumption. Each server type is profiled by running the target application, and the best server combinations for all application performance rates are then computed. Our previous work introduced the concept of a heterogeneous energy proportional infrastructure, named "Big,Medium,Little" (BML) [3]. The present work extends it by providing a scheduler that handles reconfiguration decisions, such as dynamic application migrations and management of computing resources with switch on and off actions. Such reconfiguration decisions are made considering energy proportionality while respecting QoS constraints. Depending on the characteristics and constraints of running applications, our system can be adapted by selecting the right scheduling policy and tuned to minimize the energy consumption.

The rest of this paper is organized as follows. Section II discusses the state of the art on energy proportionality in data centers. In Section III, we briefly describe our framework components, namely infrastructure and applications, as well as the modules responsible for reconfiguration decisions. We explain the target applications and their load characteristics in Section IV. Section V details the choices of heterogeneous architectures for our BML infrastructure. An implementation of the framework with real hardware and a use-case application is discussed and evaluated in Section VI. Finally, Section VII concludes the paper and discusses future work.

II. Related Work on Energy Proportionality

Barroso and Holzle [2] introduced the concept of energy proportionality by observing more than five thousand servers from a Google data center and discovering that their average utilization is between 10 and 50%. This is an issue because typical servers are not very energy efficient under low utilization. In [4], Varsamopoulos *et al.* define two metrics to quantify energy proportionality: IPR, for Ideal to Peak Ratio, which measures the dynamic power range, and LDR, for Linear Deviation Ratio, to evaluate the linearity of the consumption. They studied the evolution of energy proportionality and found that recent servers feature better characteristics, but most time it only concerns one aspect: a larger dynamic power range or an improved linearity.

Attempts have been made to improve the power management of servers, such as Running Average Power Limit (RAPL) introduced by Intel in their Sandy Bridge processors.

Via this mechanism a user can specify a power consumption threshold that the processor will not exceed during a given period. The energy savings achieved by RAPL have been evaluated facing different use-case applications, respectively data stores and latency critical workloads [5], [6]. This power capping technology offers better energy proportionality, but does not solve the problem of high idle consumption.

Nathuji *et al.* [7] have shown that resource heterogeneity can be exploited to improve energy efficiency. The same heterogeneity concept has recently been used in mobile devices. For instance, ARM big.LITTLE processor [8] consists in putting on the same board two different processors with their own power consumption and performance characteristics. The idea is to offer a processor with low power consumption that delivers low-level performance, and a more powerful and power consuming processor, to process more intensive tasks. ARM developed this technology to save mobile devices' battery life during idle periods. The concept has been adapted to server scale [9]. The authors designed a motherboard containing a server processor, called primary server, and a low power processor, called the Knight, which is always on and wakes up the primary server in case of high load.

Our work aims to achieve energy proportionality at the scale of a data center composed of heterogeneous architectures. We take advantage of heterogeneity of existing architectures by combining them with the goal of achieving energy proportionality. The strong aspect is that it does not rely on a specific processor design. This idea has been introduced previously [10], and its feasibility has been studied [3]. The present work implements a BML infrastructure with scheduling policies that take into account both benefits and drawbacks of using independent machines. Meisner *et al.* [11], exhibit the need for fast wake-up actions for servers to eliminate idle power. While we agree on this point, our work copes with existing architecture and their on/off overheads as they are, and evaluates their gains. In [12], we detailed the methodology for building energy proportional data centers with heterogeneous independent machines. This paper provides extended evaluations, results and discussions about the reconfiguration decisions and the relevance of the infrastructure.

III. Framework Overview

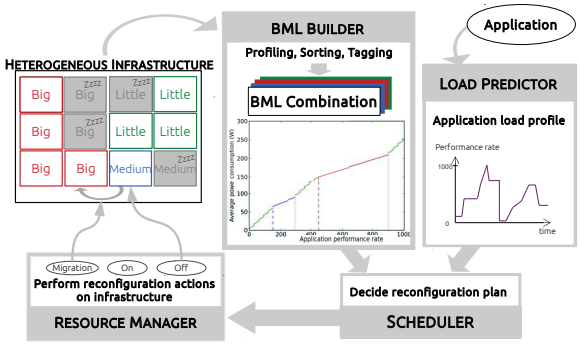


Fig. 1. Framework overview.

Figure 1 depicts our framework, its components and their interconnections. On the left-hand side, the *heterogeneous infrastructure* comprises different architecture types. Each type of architecture must be profiled regarding power consumption and performance for the target application. The *BML builder* module combines these profiles in order to find the ideal BML combination as further detailed step by step in Section V. On the right-hand side, the application and its load profile are shown. This profile may be known a-priori or predicted as discussed later in Section IV. The *load predictor* module forecasts the performance rate that an application requires at a certain point in the future. Then, the *scheduler* consults the *BML Combination* to know the ideal combination of hardware to achieve this rate. The scheduler can eventually decide to perform a reconfiguration towards this new ideal combination. If a reconfiguration is required, its execution plan is passed to the *resource manager*, which is responsible for performing all reconfiguration actions. These actions consist in switching on and off machines and migrating the application if necessary.

Although this framework is generic and each module can be implemented in various ways, we discuss an implementation and its evaluation in Section VI.

IV. Characterizing the Application and its Load

Infrastructure configuration and application parameters are factors that can affect the energy proportionality of our data center solution. A precise application characterization can enable appropriate actions to be taken to minimize the energy consumption. This section defines all application and load properties that must be determined to accomplish such goal.

Our system considers applications with variable load and it adapts the infrastructure to load conditions so that the energy consumption more closely matches resource utilization. For such, the application performance is characterized using an *application metric* that represents the amount of work performed over a given time unit. This metric is used to assess the application performance independent of the underlying architecture and to determine the QoS. As the system seeks to minimize the energy consumption without degrading the QoS, the intended quality directly impacts the relevance of reconfiguration decisions. With respect to performance, applications can be classified as *critical* when they have stringent performance requirements, and *tolerant* for applications with soft QoS requirements. *Critical* applications can be found in banking and medical areas where delays have serious consequences. More *tolerant* applications are found in, for instance, enterprise services, or services with flexible deadlines. Certain applications lie in between these classes, and hence, depending on the use-case, several intermediate classes be required.

Applications are also classified on whether they can be migrated across machines, and whether they can run on multiple architectures. The former is determined by how the application maintains state and on the amount of data to transfer. To characterize this ability we must evaluate the application's migration overhead, both in terms of duration and energy consumption. Another important characteristic concerns the

application malleability — its ability to be distributed across several machines — in which case the minimum and maximum number of instances should be specified. This criterion poses a constraint when computing the possible machine combinations for running the application.

The knowledge of how load evolves, an important parameter in our system, can be *perfect*, when the load can be determined with a certain precision; *partial*, where certain characteristics are known, such as weekly, diurnal, hourly patterns, but the accuracy of load variations is unknown; and *unknown* when no a priori information is available, and the load must be predicted for future intervals.

V. Building BML Infrastructure in 5 Steps

A first step towards designing resource management techniques that are more energy proportional consists in determining the energy consumption and performance characteristics of the hardware architectures available in a data center. Hence, we profile each machine type considering the application metric and the energy consumption. With respect to energy consumption, a machine profile contains at least two data points, namely its idle power consumption and its consumption at maximum performance. We also evaluate the overhead of switching a machine on/off, both in terms of time required and energy consumption. Once a profile is built, a computational phase is conducted to build the ideal BML combinations that achieve energy proportionality.

As we intend to plan the required capacity to achieve energy proportionality, our approach is not constrained by the number of machines of each type. We consider that enough machines of each type are available to choose from when building machine combinations. This enables creating perfect combinations based on hardware profiles. With minor changes, this work can consider cases where an heterogeneous infrastructure has already been established, and there are thus limited numbers of machines of each type.

This section lays out the multiple steps towards creating an energy proportional data center for a given application, which corresponds to the *BML builder* module we introduced in Section III. 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. We evaluate the approach and present results considering real hardware in Section VI.

A. 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. To build the profile, the power consumption of a machine type under a given performance rate, and its maximum performance rate must be determined. *Performance rate* is expressed by an application metric such as number of requests processed per second for a web server, or frame rate for video rendering. The profile of an architecture i provides:

- $idlePower_i$: average idle power of architecture i, in Watts.
- $maxPerf_i$: maximum performance rate, expressed with the application metric (*e.g.*, nb of requests processed per second).
- $maxPower_i$: average power consumed when it reaches $maxPerf_i$ rate, expressed in Watts.

Given this information and assuming that the power consumption is linear between $idlePower_i$ and $maxPower_i$, a function, named $powerFor_i$, is created to compute the power consumed by the architecture i under the specified performance rate $perfRate$. The assumption on linear power consumption might lead to a small under- or over-estimation, as it has been studied by Rivoire *et al.* in [13]. Yet, this linear approximation is precise enough for our solution, and it eases the profiling and profile building phases. Although acquiring more intermediate data points, if the application allows, or considering potential DVFS (Dynamic Voltage and Frequency Scaling) system, would enable more precise profiles, our methodology would not be affected by this higher precision.

B. Step 2: Sort Architectures to Keep Only BML Candidates

Building a BML infrastructure starts by sorting machines by decreasing maximum performance. 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, then it is removed from the BML candidates as it does not respect the required properties to improve energy proportionality. A list with the relevant architectures for building a BML infrastructure will be available at end of this step.

Figure 2 illustrates the profiles of four architectures A, B, C, and D. Beyond the point $(maxPerf_i, maxPower_i)$ of each architecture, its profile is repeated to picture multiple nodes. After the execution of step 2, only three architectures are kept as good BML candidates. Architecture D is discarded because its maximum power consumption is greater than A's, which is the most powerful machine, and architectures B and C both have lower idle consumption. Once this filtering is complete, architectures are sorted and labeled *Big*, *Medium* or *Little* according to their performance. Here the result is: A $\leftarrow Big$, B $\leftarrow Medium$, and C $\leftarrow Little$.

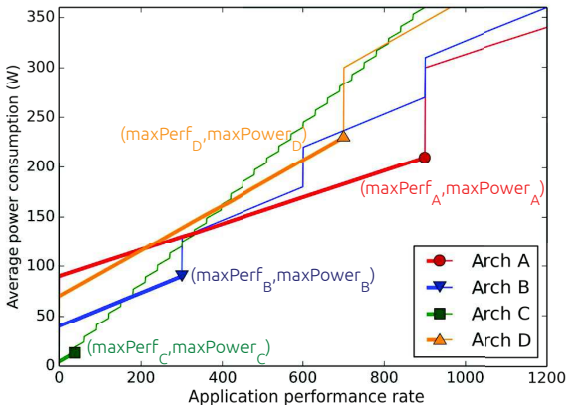


Fig. 2. Architectures A, B and C are good candidates for BML infrastructure, but Architecture D will be removed from consideration due to its poor energy efficiency compared to other architectures.

C. Step 3: Finding Crossing Points between Architectures

Algorithm 1 $CrossPoints_{step1}$: Finds crossing points between architectures

Input: BML candidates: BML
Output: Crossing points: $crossPts$, Updated list: BML

$LMB \leftarrow BML.reverse()$
$crossPts \leftarrow [\,]$
$j \leftarrow 1$
for $i \in [0, LMB.length - 2]$ **do**
  $current \leftarrow LMB[i]$
  $next \leftarrow LMB[i+1]$
  while $j \leq maxPerf_{next}$
    and $powerFor_{current}(j) < powerFor_{next}(j)$ **do**
    $j \leftarrow j + 1$
  end while
  $crossPts.append(j)$
  $minThreshold_{next} \leftarrow j$
end for

This step determines how chosen architectures should be combined to create the most power-proportional infrastructure. We define the minimum utilization threshold for each architecture considering 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 performance rate becomes more relevant than i's when considering power consumption. Initially, all minimum thresholds are set to 1. Whilst this threshold will remain 1 for the *Little* architecture, the function described in Algorithm 1 will recompute the thresholds for all remaining architectures. The points where an architecture becomes preferable over another are termed as *crossing points* as they represent the points where power profiles meet.

Left part of figure 3 illustrates this step with architectures A, B and C, now denoted Big, $Medium$ and $Little$. The utilization 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, and next step will improve it.

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

The previous step computes the crossing points between homogeneous combinations of machines, but with three architectures or more, one must determine whether adding $Little$ nodes to $Medium$ combinations help improve power proportionality and reduce the gap between $Medium$ and Big architectures. The function, detailed in Algorithm 2, re-evaluates the computed crossing points between $Little$ and $Medium$, except for the first one, that cannot be questioned as $Little$ is the smallest architecture of the infrastructure. Right part of figure 3 shows that minimum threshold of Big architecture is updated after this step.

This algorithm calls the function $idealBML$ described in Algorithm 3, 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 in the computed combination. A last phase, performed at the end of Algorithm 2, 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 $maxPerf_i$, it means that the utilization range of this architecture is empty, and thus it must be removed from the infrastructure. Under such case, algorithms 1 and 2 should be executed again with the updated list of BML candidates.

Algorithm 2 $CrossPoints_{Step2}$: Finds crossing points between architectures and combinations of small architectures

Input: BML list: BML, Crossing points: $crossPts$
Output: Updated BML list: BML

for $i \in [1, crossPts.length - 1]$ **do**
  $current \leftarrow LMB[i]$
  $next \leftarrow LMB[i+1]$
  if $(crossPts[i] - 1 \;\%\; maxPerf_{current}) == 0$ **then**
    $j \leftarrow crossPts[i]$
    $baseLvl \leftarrow crossPts[i] - 1$
    $c, power \leftarrow idealBML(j - baseLvl)$
    while $j \leq maxPerf_{next}$ **and**
      $power < powerFor_{next}(j - baseLvl)$ **do**
      $j \leftarrow j + 1$
      $c, power \leftarrow idealBML(j - baseLvl)$
    end while
    $minThreshold_{next} \leftarrow j$
  end if
end for

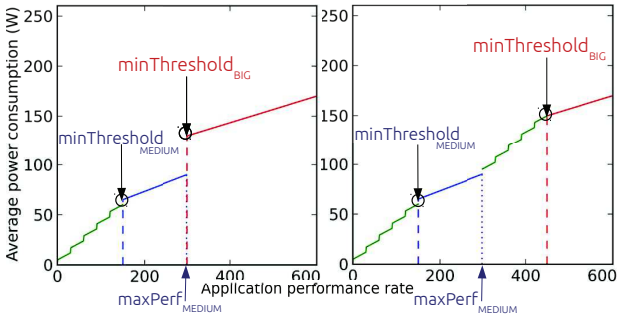


Fig. 3. On left: *Step 3* - First step of crossing points computation between $Little$ and $Medium$, and between $Medium$ and Big. On right: *Step 4* - Second step of crossing points computation between $Little$ and $Medium$, and between combinations of $Medium - Little$ and Big

E. Final step: Computing Ideal BML Combination

Algorithm 3 details the function that computes the ideal machine combinations and their corresponding power consumption in order to achieve a given performance rate. Building BML combinations is similar to a bin-packing problem where 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 – the target performance – but it can be divided into as many pieces as necessary, and of any size. 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.

Algorithm 3 $idealBML$: Computes ideal BML combination and its instantaneous power consumption for $perfRate$

Input: BML list: BML, Performance rate: $perfRate$
Output: BML combination: $combination$, Power consumption of combination: $power$

$combination \leftarrow [\,]$
$power \leftarrow 0$
$remRate \leftarrow perfRate$
for $arch \in BML$ **do**
  $nb \leftarrow$ int $(remRate/maxPerf_{arch})$
  $remRate \leftarrow remRate - nb \times maxPerf_{arch}$
  $power \leftarrow power + nb \times maxPower_{arch}$
  if $remRate \geq minThreshold_{arch}$ **then**
    $nb \leftarrow nb + 1$
    $power \leftarrow power + powerFor_{arch}(remRate)$
    $remRate \leftarrow 0$
  end if
  $combination.append(nb)$
end for
return $combination$, $power$

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.

VI. Experimental Evaluation

A. Experimental Setup and Profiling Results

Energy proportionality can be approached by minimizing energy consumption during periods of low load, which can be achieved with hardware consuming little energy when idle. ARM processors were originally designed for embedded devices to extend battery life. Their performance has improved over time, however, they are not yet as powerful as traditional data center servers. Existing work has studied the use of low power processors for handling data center workloads like web servers [14] and big data [15]. It has been shown that popular Raspberry Pi's are efficient for hosting static web servers while consuming less power than a standard server [14]. ARM processors also offer interesting consumption-performance ratios for database query processing compared to an Intel Xeon processor [15]. On the other hand, these studies show performance limitations, concluding that these equipments cannot compete with standard servers for more demanding workloads such as dynamic web servers or I/O intensive big data applications. This motivated us to combine both low power processors and regular servers for building a heterogeneous data center.

Paravance: x86 Intel Xeon E5-2630v3 (2x8 cores); *Taurus*: x86 Intel Xeon E5-2630 (2x6 cores); *Graphene*: x86 Intel Xeon X3440 (1x4 cores); *Chromebook*: ARM Cortex-A15 (1x2 cores); *Raspberry*: ARM Cortex-A7 (1x4 cores). A *WattsUp?Pro* wattmeter monitors power consumption of the *Samsung Chromebook* and *Raspberry Pi2B+*. The x86 servers are available at *Grid'5000* [16], a French experimental testbed for research, whith power monitoring data accessible via *Kwapi* [17].

A stateless web server is our target application because:
• a load balancer could allow the load to be distributed among several web server instances;
• being stateless, an application can be easily migrated as it consists in stopping a server instance and launching a new one on the destination machine, and updating the load balancer;
• it is a perfect example of application with variable load over time, and its performance can be characterized with an application metric: number of requests processed per second.

We use *lighttpd* as web server and *siege* as web benchmark tool. he content of the web server is a python cgi script. Each request consists in a loop of random number generation, while loop iterations is also chosen randomly between 1000 and 2000. The request response is a static html page containing this later integer. We execute the benchmark with an increasing number of concurrent clients in order to find the maximum request rate that can be processed. Each test runs for 30 seconds and the maximum performance is the average of 5 results. We also measure On/Off durations and energy consumption. Table I and Figure 4 present the results.

TABLE I
Performance and Power Profiles of Each Architecture.

Architecture Codename	MaxPerf (reqs/s)	Idle-Max Power (Watts)	On_t (s)	On_E (Joules)	Off_t (s)	Off_E (Joules)
Paravance	1331	69.9 - 200.5	189	21341	10	657
Taurus	860	95.8 - 223.7	164	20628	11	1173
Graphene	272	47.7 - 123.8	71	4940	16	760
Chromebook	33	4 - 7.6	12	49.3	21	77.6
Raspberry	9	3.1 - 3.7	16	40.5	14	36.2

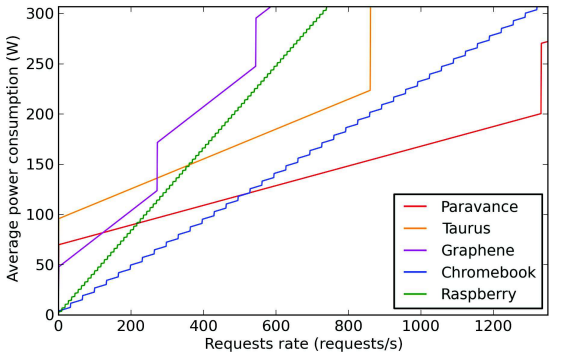
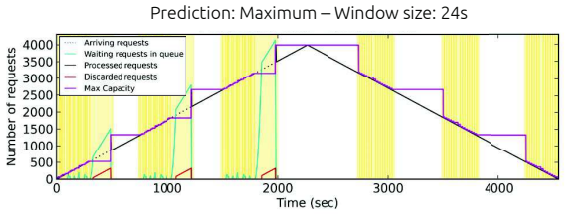


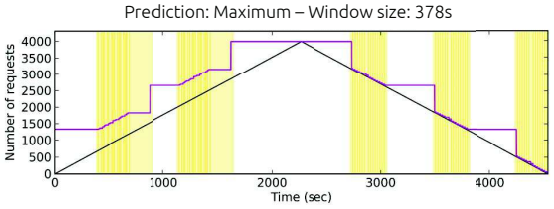
Fig. 4. Power and performance profiles of web servers acquired from experiments on 5 different architectures

B. Results at Server-Scale

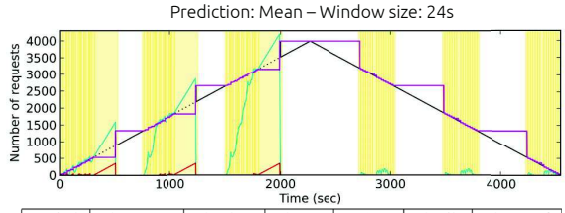
Figure 4, result of *Step 1*, shows the profiles acquired experimentally. After *Step 2* which consists in sorting and tagging architectures according to their maximum performance and power consumption, *Taurus* machine is removed from the infrastructure as its maximum power consumption is higher than *Paravance*'s (223.7 W against 200.5 W) while



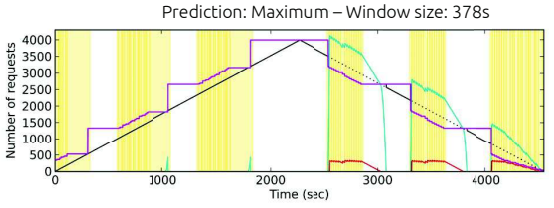
Period	%Process.	%Delay.	%Lost	JpR	%Util.	nbReconf
Up	98.15	19.66	1.85	0.1783	94.64	51
Down	100	0	0	0.1638	85.86	44
Total	**99.08**	**9.74**	**0.92**	**0.1710**	**90.25**	**95**



Period	%Process.	%Delay.	%Lost	JpR	%Util.	nbReconf
Up	100	0	0	0.1840	67.85	34
Down	100	0	0	0.1638	85.86	44
Total	**100**	**0**	**0**	**0.1739**	**76.85**	**78**



Period	%Process.	%Delay.	%Lost	JpR	%Util.	nbReconf
Up	97.60	45.45	2.40	0.1782	95.98	51
Down	100	0.86	0	0.1633	87.93	45
Total	**98.80**	**22.88**	**1.20**	**0.1707**	**91.95**	**96**



Period	%Process.	%Delay.	%Lost	JpR	%Util.	nbReconf
Up	100	0.15	0	0.1830	79.26	41
Down	93.01	54.47	6.99	0.1606	95.49	51
Total	**96.5**	**26.33**	**3.5**	**0.1722**	**87.38**	**92**

Fig. 5. Comparison of different policy settings on the same input trace consisting of regular slope up and down phases.

delivering lower performance (860 requests/sec against 1331). The remaining architectures are tagged as follows: *Paravance* $\leftarrow Big$, *Graphene* $\leftarrow Medium1$, *Chromebook* $\leftarrow Medium2$, *Raspberry* $\leftarrow Little$. Execution of *Step 3*, which computes the crossing points between architectures, shows that the profile of *Graphene* ($Medium1$) never crosses any other architecture's profile, thus it is removed from the candidates. Our final heterogeneous infrastructure comprises *Paravance* (Big), *Chromebook* ($Medium$) and *Raspberry* ($Little$). Their minimum utilization thresholds are 1 request per second (requests/s) for $Little$, 10 requests/s for $Medium$ and 529 requests/s for Big. The ideal BML combination, the result of *Final Step*, is depicted in Figure 6. Big architecture's profile is also represented in Figure 6 in order to demonstrate the gains of the heterogeneous combination. In addition, we introduce $BML\ linear$ architecture 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 how a 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.

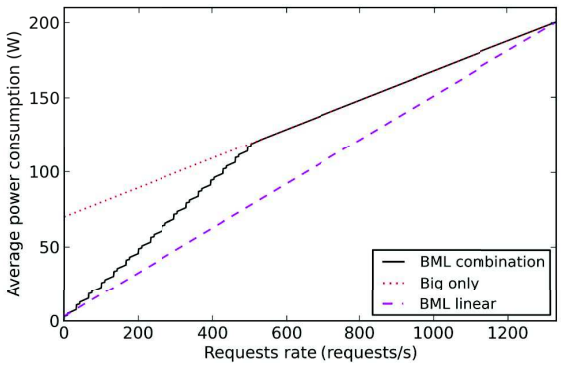


Fig. 6. Consumption of BML combination over an increasing performance rate, until $maxPerf_{Big}$, compared to Big and $BML\ linear$

C. Finding the Best Policy Settings

For evaluations, we developed a simulator in Python, which takes as inputs the hardware profiles and a trace file describing the application load variations; hence we consider the actual load known throughout the experiment. We emulate load prediction with a sliding time window of the future load values. Two approaches are used: computing the mean of the window values, and picking the greatest value. Two different window sizes are considered: 24 and 378 seconds, corresponding respectively to 2 times the shortest and 2 times the longest switch on duration. The predicted load value is used to compute the BML combination. The scheduling policy is pro-active; at each prediction leading to a new hardware combination, a decision of reconfiguration is taken. During the reconfiguration, no other decision can be made to ensure the completion of on and off actions. The next window for prediction starts from the reconfiguration completion time. When the prediction results in no combination changes, the window just slides one time step forward, a second in this case. If requests that arrive at a time step cannot be processed immediately due to under-provisioning, they are put in a waiting queue. Then, at each time step, already waiting requests take priority over requests that have just arrived. To avoid starvation, requests wait for maximum 2 seconds before being discarded. To understand the behavior of the framework, a simple trace is generated consisting in one up and one down phase of the same length,

starting from 0 to 3993 requests/second, corresponding to 3 times the maximum processing capacity of Big architecture.

Figure 5 presents simulation results for the four different settings. It shows 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 request (JpR), infrastructure utilization and number of reconfiguration decisions. These metrics are computed separately for the up and down phases (from 0 to 2268s and from 2269 to 4536s), and on the whole duration. Each graph represents the temporal evolution of the simulation: arriving requests, maximum processing capacity of current combination, waiting and discarded requests over time. Vertical lines correspond to reconfigurations periods.

We observe that the system behaves differently during up and down phases due to heterogeneous durations of on and off actions; hence the reason why two window sizes are considered. The long window is very accurate for the ascending phase as switch on actions take more time and need to be anticipated. But the mean prediction on the long window is not accurate for the descending phase because it predicts load decrease too early for the immediate effects of switch off actions. Based on these results, we conclude that the maximum prediction over the long window is the best choice for *critical* applications as it leads to no delayed and no discarded requests. If the application is *fault tolerant*, then the choice is less constrained. The setting that provides the best results in terms of joules per request (0.1707 JpR) and percentage of utilization (91.95%) while discarding 1.2% requests, is the mean prediction based on the short window.

D. Big Only, Big-Medium or BML?

We compared BML against infrastructures comprising either only Big machines or both Big and $Medium$ nodes. We evaluate the three scenarios with the 1998 World Cup website access logs [18]. The 48th day is chosen due to the large number of requests; mean rate is 566 requests/second, with a peak of 1867 requests/second. We choose the policy settings as though the application was critical, meaning a maximum prediction over a long window of 378 seconds. This allows to compare the results with the same number of total processed requests. Figures 7, 8 and 9 show the temporal evolution for respectively Big, Big-Medium and BML cases. Apart from processed requests and maximum capacity of the infrastructure, we show the number of each type of machines over time, and vertical lines corresponding to reconfigurations periods.

We see that heterogeneity allows to adapt the infrastructure at finer grain, and consequently leads to higher utilization: 69,7% for BML, 69% for BM against 40.7% for Big only. The energy waste is reduced especially during periods of low load like the first part of the day. Similar conclusion can be drawn from the joules per request metric: 0.2155 JpR for BML, 0.2157 JpR for BM against 0.2268 JpR for Big only. The small difference between BML and BM infrastructure stems from the fact that our BML combination is not optimal as explained in Section VI-B. $Little$ architecture has a small utilization range of only 9 requests, which explains why its presence cannot bring significant improvements. Nevertheless, we observe that despite a higher number of reconfigurations in BML case compared to Big-Medium, exactly 65 more, the joules per request results are better. We still benefit from more heterogeneity even if the utilization range of the additional architecture is small. Ideally, it would be preferable to have ranges of utilization of same length for all architectures.

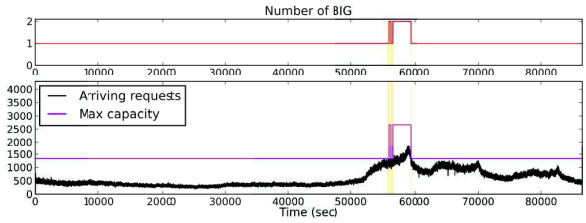


Fig. 7. Big only - JpR: **0.2268**, Utilization: **40.7%**, NbReconf: **4**

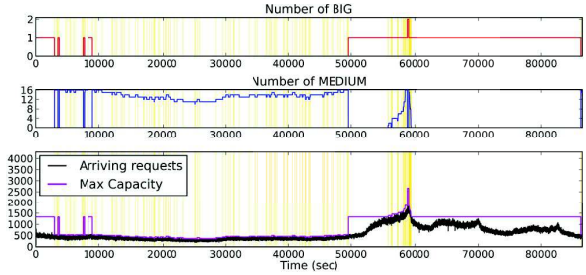


Fig. 8. Big-Medium - JpR: **0.2157**, Utilization: **69%**, NbReconf: **129**

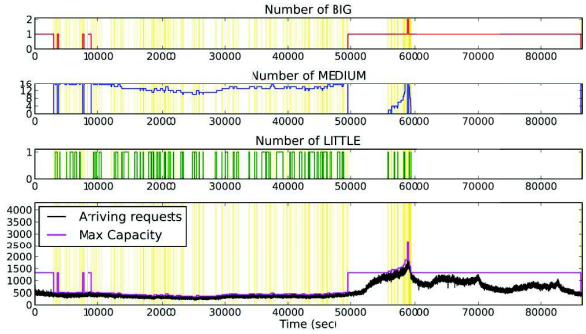


Fig. 9. Big-Medium-Little - JpR: **0.2155**, Util.: **69.7%**, NbReconf: **194**

E. Comparison with Lower and Upper Bounds

We compared BML infrastructure and placement algorithm against a theoretical BML lower bound and two homogeneous upper bounds. We run the simulations for days 6 to 92 of 1998 World Cup traces [18]. The resulting scenarios are as follows:

- *UpperBound_Global*: a data center with a constant number of homogeneous *Big* servers, computed according to the maximum request rate: 4089 requests/s in day 73 (4 *Big* machines) This is an example of a classical over-provisioned data center.
- *UpperBound_PerDay*: a data center with homogeneous *Big* servers dimensioned each day according to the daily maximum rate. This is an example of coarse grain capacity planning.
- *Big-Medium-Little*: our BML infrastructure and placement algorithm with same policy setting as for Figure 9: maximum

prediction and long window. The total consumption per day contains the energy consumed by computation and the energy from on/off reconfigurations made during the day.

• *LowerBound_Theoretical*: the minimum computing energy achieved with our BML infrastructure if the data center is dimensioned each second with the ideal BML combination. This is an unreachable lower bound considering no on/off latency and no on/off energy costs.

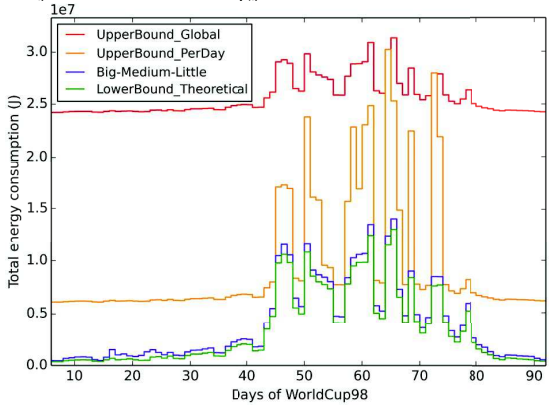


Fig. 10. Energy consumption comparison with lower and upper bounds.

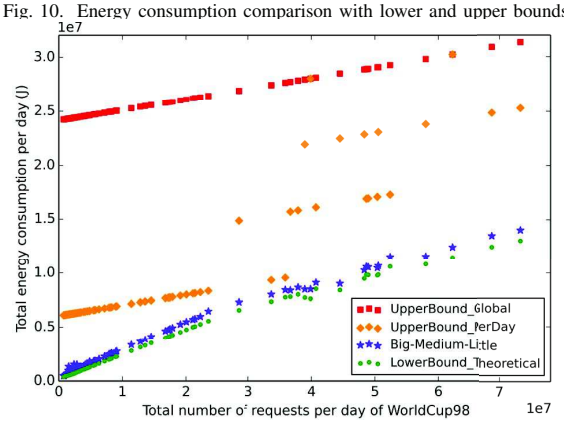


Fig. 11. Energy proportionality comparison with lower and upper bounds

Figure 10 summarizes the results. Our *Big-Medium-Little* solution is very close to the theoretical lower bound. On average over these 86 days, BML solution consumes 32% more than the lower bound, minimum being 6.8% for day 52 and maximum 161.4% for day 23. The graph demonstrates the high static costs coming from classical over-provisioned data centes, and allows to see the objective reached with our solution that is an energy consumption more proportional to the actual daily load. This is clearly noticeable on Figure 11, 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 lower bound proves the relevance of the reconfigurations as energy consumed by on and off switches does not represent a significant overhead.

VII. Conclusion and Perspectives

We proposed a methodology to reach energy proportionality by designing a data center composed of heterogeneous architectures. We demonstrated its feasibility with existing hardware and evaluated its performances when hosting stateless web servers with variable load. We detailed how the scheduler takes reconfiguration decisions to meet different QoS constraints while minimizing energy consumption. We showed the benefits of our BML solution compared to homogeneous ones and against classical data center management and proved that we drastically reduce static costs and achieve energy proportionality. As future work we will investigate the impact of load prediction. It is also worth considering other hardware combinations than pre-computed BML combinations as reconfiguration possibilities, and consider their time and energy overheads in the decision process.

Acknowledgments

Experiments were carried out using the Grid5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities and organizations (https://www.grid5000.fr). This research is partially supported by the French ANR DATAZERO project, and European CHIST-ERA STAR project.

References

[1] Uptime Institute. 2014 Data Center Industry Survey.
[2] L.A. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 2007.
[3] V. Villebonnet, G. Da Costa, L. Lefevre, J-M. Pierson, and P. Stolf. "Big, Medium, Little": Reaching Energy Proportionality with Heterogeneous Computing Scheduler. *Parallel Processing Letters*, 25(3), 2015.
[4] 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.
[5] 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.
[6] D. Lo, L. Cheng, R. Govindaraju, L.A. Barroso, and C. Kozyrakis. Towards Energy Proportionality for Large-scale Latency-critical Workloads. *SIGARCH Comput. Archit. News*, 2014.
[7] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting Platform Heterogeneity for Power Efficient Data Centers. In *International Conference on Autonomic Computing (ICAC'07)*, 2007.
[8] ARM big.LITTLE: The Future of Mobile. *ARM White Paper*, 2013.
[9] D. Wong and M. Annavaram. Scaling the Energy Proportionality Wall with KnightShift. *IEEE Micro*, 2013.
[10] G. Da Costa. Heterogeneity: The Key to Achieve Power-Proportional Computing. In *IEEE CCGrid*, 2013.
[11] D. Meisner, B.T. Gold, and T.F. Wenisch. PowerNap: Eliminating Server Idle Power. *SIGARCH Computer Architecture News*, 2009.
[12] V. Villebonnet, G. Da Costa, L. Lefevre, J-M. Pierson, and P. Stolf. Dynamically Building Energy Proportional Data Centers with Heterogeneous Computing Resources. In *IEEE Cluster (short paper)*, 2016.
[13] S. Rivoire, P. Ranganathan, and C. Kozyrakis. A Comparison of High-level Full-system Power Models. In *Conference on Power Aware Computing and Systems*, HotPower'08. USENIX Association, 2008.
[14] 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.
[15] D. Loghin, B.M. Tudor, H. Zhang, B.C. Ooi, and Y.M. Teo. A Performance Study of Big Data on Small Nodes. *VLDB*, 2015.
[16] R. Bolze et al. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed. *Journal of HPC Applications*, 2006.
[17] 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.
[18] WorldCup'98 Traces. http://ita.ee.lbl.gov/html/contrib/WorldCup.html.