Big, Medium, Little: Reaching Energy Proportionality with Heterogeneous Computing Scheduler

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

***Abstract*—Energy savings are among the most important topics concerning Cloud and HPC infrastructures nowadays. Servers consume a large amount of energy, even when their computing power is not fully utilized. These static costs represent quite a concern, mostly because many datacenter managers are over-provisioning the infrastructure compared to the actual needs. This results in a high part of wasted power consumption. In this paper, we proposed the BML (Big, Medium, Little) infrastructure, composed of heterogeneous architectures, and a scheduling framework dealing with energy proportionality. We introduce heterogeneous power processors inside datacenters as a way to reduce energy consumption when processing variable workload. Our framework brings an intelligent utilization of our infrastructure, by dynamically execute applications on the architecture that suits their needs, while minimizing energy consumption. In this paper we focus on stateless web servers, which imply less technical challenges, and we show the achieved energy savings thanks to our scheduling policy exploiting the heterogeneity of the platform.**

Keywords*—*Energy proportionality, Heterogeneous hardware, Applications profiling

I. Introduction

In [1], authors estimate that worldwide datacenters consumed up to 270 TWh in 2012, which accounts for almost 2% of global energy consumption. Therefore electricity consumption and its cost is one of the main limitation for building such infrastructures. But besides this economical point of view, the ecological aspect must also be considered. This huge energy consumption has a relevant impact on our environment due to the resulting CO_2 emissions. According to several weather agencies, 2014 is likely to be the warmest year on record. This fact proves that it is crucial to make efforts to increase energy efficiency of these infrastructures on which we rely more and more.

Architectural designs of these large scale datacenters are far from perfect because not all the consumed energy goes to computing. This is highlighted by the PUE metric, which stands for "Power Usage Effectiveness" promoted by The Green Grid in 2007 [2]. This measure is a ratio of the total energy consumed by the whole datacenter upon 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. Despite all the controversy about how companies compute and use this metric, the PUE does not show the real efficiency of computing equipments. Indeed inside a datacenter, servers are in most cases always powered on even if they are not doing computation. In this situation the energy is effectively consumed by IT equipment but is completely wasted. The problem is that when a server is idle (powered on but without activity), its energy consumption is already significant. Some idle servers can consume as high as 50% of its maximum power consumption when fully loaded. In addition, people usually only focus on the maximum energy consumption of a datacenter, but not enough to 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 [3]. They conducted experiments in a Google datacenter, and noticed that servers are mostly used 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 energy consumed when a machine is idle is called the static consumption and this is the issue we want to tackle in our work. For example on Fig.1 from [3], the static consumption represents 50% of the peak. Our objective is to reduce this static cost as much as possible, closest to zero, in order to have 100% of dynamic consumption. They have named this goal "Energy Proportional Computing". An architecture with such a consumption pattern would bring significant energy savings.

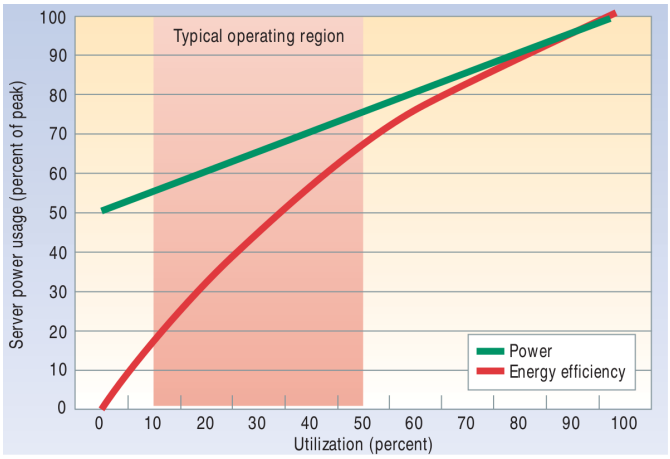


Fig. 1. Server power consumption and energy efficiency from 0 to 100% utilization (Figure from [3])

To get closer to energy proportionality, we propose the BML infrastructure that is composed of heterogeneous computing resources. Around this infrastructure we propose the BML framework which gathers several modules for applications and resources intelligent management, a well as the global scheduler. We firstly show how important are the experimental measurements and profiling of the hardware in order to calibrate the framework. Then we choose to focus on the scheduler module and show the gains in energy consumption of our proposed BML combination of heterogeneous resources considering stateless web servers with highly variable loads.

This paper begins with an overview of related works on energy efficiency and proportionality in next section. Our proposed BML infrastructure architecture is introduced in section III. We detail the technical challenges brought by heterogeneity in section IV, as well as the proposed solutions to tackle them and the chosen technologies. In section VI, we validate our framework, with a focus on stateless applications and explain our simulation scenarios together with the obtained results. Finally we discuss about the current limitations of our approach, draw some conclusions and propose some perspectives for future work.

II. Related work

Energy savings in clouds and HPC infrastructures is a popular and quite recent research field. Many works have been done in this area to find solutions to optimize the utilization of resources inside datacenters. A famous approach is consolidation, which consists in gathering the working load on the fewest number of servers to be able to switch off the unused ones. This can be achieved thanks to a key technology which is virtualization. It allows several independent operating systems to coexist on a single physical machine. Live migration [4] is the mechanism used to dynamically move virtual machines through physical servers without impacting applications running inside. Performing consolidation aims at saving energy by freeing lightly loaded machines. The goal is to switch unused servers off, or put them in a low power mode, and only turn them on when they are needed. 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. Most consolidation approaches are based on heuristics algorithms, which are variants of the bin packing problem, but other alternatives have been proposed and tested such as constraint programing [5], genetic algorithms or Ant-Colony metaheuristics [6]. Another green leverage is DVFS, which stands for "Dynamic Voltage and Frequency Scaling". The principle is to adapt the frequency of the processor to the current server needs, because the energy consumption decreases when the frequency is reduced. However like other leverages, important energy savings can only be reached if those actions are performed wisely, and this will rely on a good knowledge or prediction model of the workload. In [7], the authors propose to monitor performance counters in order to get current profiles of running systems and predict their evolutions. This system allows to take decisions according to the predictions and thus make more effective energy savings.

But these approaches have some limitations, they only reduce the overall energy consumption. Consolidation enables the servers to be fully loaded, where they are the most efficient, but the problem of high static consumption remains. With our work we want to bring a solution which eliminates static costs by trying to reach energy proportionality. The goal is to approach a nearly null consumption at idle state, and then a linear consumption proportional to the load. If such a proportional hardware, or system, could exist, then consolidation would not necessarily be needed because the energy efficiency of the system would be constant.

Regarding proportional computing, some works [8] [9], propose metrics to evaluate the proportionality of an architecture. The first one compares the consumption curve as a function of load, to the ideal proportional linear curve. While the second one defines two separate metrics: one to measure the difference between idle and maximum consumption, and another to measure the linearity of this consumption. These metrics are then applied to existing architectures to study the evolution of the hardware from this point of view through recent years. The architectures are in general more and more proportional, but it is noted that meanwhile the gap between maximum and idle consumption is reduced, the linearity is degraded. In [10], authors are exploiting a quite new technology introduced by Intel in their Sandy Bridge processors which is called RAPL, standing for "Running Average Power Limit". This feature allows the users to specify an average limit for power consumption of the processor over a given period of time. The system then automatically regulates its behavior to fit its consumption under the limit. This technology offers better than DVFS because it can be controlled with a finer grain. Energy efficiency can be enhanced but perfect proportionality is not reached yet and still seems far away.

One suggestion to get closer to the goal of energy proportionality is to use several architectures with different performance and consumption characteristics. This is the concept used in heterogeneous multi-core processors. Different companies have proposed their implementation of this concept like ARM with their big.LITTLE processor [11] which combines a low-power processor with a high-performance one, or Nvidia with their new Tegra K1 processor [12] that couples ARM processors with GPU accelerators. Inside those systems, the applications are chosen to run on the processor that best suit their computing needs. Moreover, they feature a shared cache memory thanks to a cache coherence interconnect system that eases the migration of tasks between the processors.

Fig.2 depicts the architecture of big.LITTLE heterogeneous processors according to ARM itself. On the left, Cortex-A15 plays as the 'big' processor and on the right Cortex-A7 is the 'little' one. The particularity of this proposition resides in the CCI module, functioning with interruptions, which brings full coherency between the two processors. This concept allows a nearly transparent task migration from one processor to another, and this enables to better fit to the evolutions of application resource needs. ARM proposes different forms of utilization of this architecture : CPU Migration and Global Task Scheduling. In the first one, each big core is paired with a little one, and only one core of each couple can be active at a time. Whereas in the second form, all cores are viewed in a global way and any core can be active or shut down independently. The last option offers more flexibility but also more complexity and brings many challenges for the system

management.

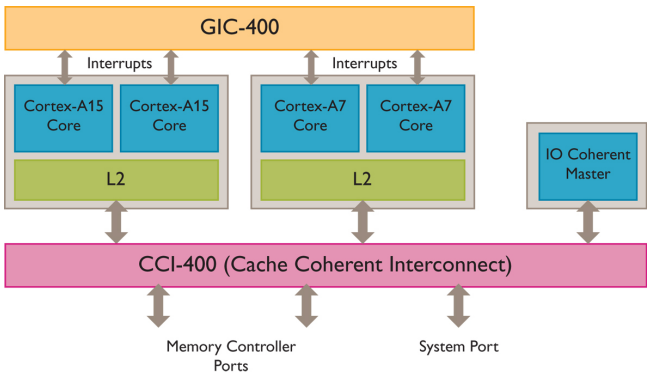


Fig. 2. big.LITTLE system architecture (figure from ARM white paper [11])

Heterogeneous multiprocessor is an innovative approach, but for the moment they are dedicated to mobile devices. The idea behind this concept is to extend battery life duration for mobile devices, so to consume as little as possible during idle periods, while delivering good performances when needed, for instance for game playing or video watching. Nevertheless, those performances are far from the ones of regular servers used inside datacenters. Indeed to reach the goal we have set, it seems that we need to mix both types of processors. This concept which is the basis of our work has been in introduced in [13]. It shows the potential benefits of having a set of heterogeneous hardware composed of Raspberry Pi, Intel Atom and Intel i7, hosting stateless web servers. The energy consumption curve gets closer to proportionality, and energy gains are important, especially for little load. Our main objective is to pursue this proposition, study how it is possible to extend it to a larger range of applications. Moreover we want to improve results of [13] by adding management of resources with switch on and off and take into account the associated overhead.

III. BML: Big,Medium,Little Infrastructure

Our basic idea is to go beyond the concept of ARM big.LITTLE by extending it to the datacenter scale. Our work is inspired by this idea, but we are adding more flexibility and using a wider variety of heterogeneous processors, as introduced in [14]. We want to exploit ARM processors with low power consumption when the load is low and keep the traditional servers for the performance. Although our idea is inspired from this concept, the two approaches differ on some points. As described in section II, ARM big.LITTLE is a multi-core processors architecture that combines two different kinds of processors. Both processors, ARM Cortex-A15 and Cortex-A7, are based on the same ARMv7-A Instruction Set Architecture (ISA). The heterogeneity in this case is relative because the difference only resides in the computing power and power consumption. To generalize this concept to large scale environments, we have to broaden the range of processors and thus having the same ISA is no longer possible. This is why the infrastructure we propose is composed of heterogeneous computing resources, where the heterogeneity concerns the architecture itself.

A. Infrastructure architecture

On Fig.3 is schematized our proposed infrastructure. In this example we consider three different types of machines that we name 'Little', 'Medium' and 'Big' in the same spirit of our inspiration model ARM big.LITTLE. Of course we do not consider three as the limit number of architectures, and this model can be extended for as many as relevant architectures. We imagine having several nodes of each type, that the scheduler can access, control and monitor in a total independent way. Our goal is to always execute applications on the most suitable architecture at any time. The most suitable architecture is defined as the one that consumes the least for the current performance needed by the application. Naturally, the performance requirements of the application may evolve over time, so the framework should be able to transfer its execution to another architecture. For instance if the CPU load demand decreases, the application should be migrated to a less powerful and less consuming architecture in order to save energy, but if it increases, the application must be transferred to a more powerful architecture in order to satisfy its needs and not impact negatively on its execution. This mechanism is represented by the "Live migration" arrows on the figure. We consider that these migrations can occur between any resources of our infrastructure and in both directions. When the machines are not utilized by any application, they are switched off or put in a sleep mode by the scheduler. On the figure, the sleeping nodes are represented in gray with a "Zzzz" sign to show their unavailability. We assume that when a machine is unavailable, its energy consumption is negligible.

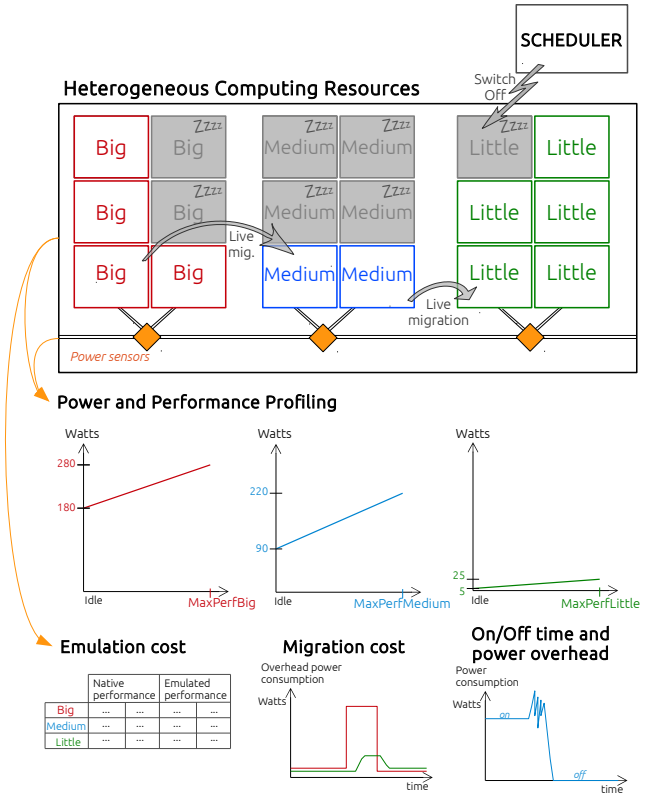


Fig. 3. Architecture of our energy proportional infrastructure

An essential first step to lead this infrastructure towards energy proportionality is profiling of the hardware. On lower part of Fig.3, we draw as example graphs and table for each aspect of the infrastructure we are profiling. As energy is our focus, computing resources must be monitored with power meters to know perfectly the consuming pattern of each machine type. These power profiles coupled with some performance profiles give a complete understanding of the behavior of each part of our infrastructure. This behavior may vary according to the type of applications used for performance profiling, that is why several profiles can be created. All these profiles are the keys to our scheduler as they characterize the heterogeneous performances and scheduling decisions are based on them. Apart from performance and power profiles of the hardware, other aspects should be investigated such as the overhead of emulation and costs for live migration and switch on/off, both concerning time and energy consumption. Without knowledge of all these parameters, scheduling decisions can not be optimal.

Heterogeneous computing brings technical challenges. In the case of ARM big.LITTLE, where the processors are totally compatible, a system of shared memory allows to easily migrate threads. On the contrary when the heterogeneity is at the architecture level, a more complex system should be found to migrate applications. Following is a study of different virtualization solutions which justifies the technologies we have chosen to implement. The implementation is then detailed in section IV.

B. Virtualization and emulation technologies

Nowadays, datacenters are mostly composed of x86 processor based servers. Since the 2000s, almost all these processors, built by Intel and AMD, have 64 bits memory addressing. They have a good performance over price ratio, and are the most widespread. However, their main drawback is their high power consumption in idle state. We consequently focus on very low consumption processors to see if we can counteract these static costs. It appears that ARM processors offer the best compromise between performance and power consumption. As a matter of fact, ARM processors are historically designed for embedded systems so the low power consumption was the main constraint. But now they are more and more designed for mobile devices such as smartphones and tablets, thus they are becoming more and more powerful. In addition, some of them recently include virtualization extensions. This last point has strengthened our idea to bring those processors into a datacenter. Furthermore, some manufacturers started lately to draw their attention towards ARM processors for server purpose. It is the case of HP with their so called "Moonshot Project" [15]. This servers contains Calxeda SoCs equiped with ARM Cortex-A9 processors. Their targets are mainly highly parallel workloads, or front-end servers with little request processing, and their goals are to reduce energy consumption but also reduce room space and costs of ownership.

Our first concern is to study the existing virtualization solutions and find if some of them are compatible with both ARM and x86 architectures, and if they can be used to perform live migration, or have a mechanism of checkpoint/restart. We also want to study other specifications such as operating systems, kernel versions, to see which solution is the least restrictive. Our objective is to select a technology upon these criteria, which will be a good basis to develop an extended migration that works between heterogeneous architectures.

We consider two main categories : virtual machines and application containers. We focus on open source solutions, that is why we selected KVM and Xen hypervisors for the virtual machine approach, and LXC and OpenVZ for containers.

TABLE I. COMPARISON OF VIRTUALIZATION VS CONTAINERS CAPABILITIES

	Virtual machines		Linux 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 not on ARM
Guest OS	any	any	only Linux based	only Linux based

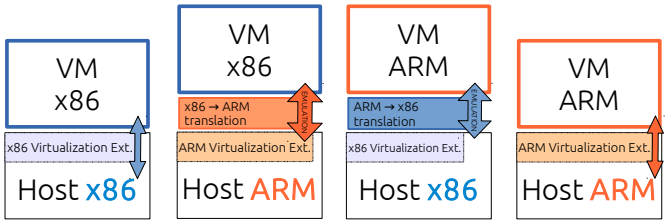
Although application containers seem to be a promising technology with a very light virtualization process and then a very low overhead, it implies many constraints. Linux containers only work with Linux based OS, and the guest shares the same operating system as well as the same kernel version as the host. Moreover we observe that checkpointing for containers is still a feature in development whereas live migration is well implemented in hypervisors like Xen or KVM. OpenVZ has a functional live migration but it works only on x86 hosts. As far as LXC is concerned, developers are not planning to implement any kind of live migration, but some work is done about checkpoint and restart of LXC containers inside the CRIU project [16] - which stands for Checkpoint/Restore In Userspace. This comparison leads us to select the virtual machine solution as it is the most common approach in datacenters and also the most general solution as it does not impose any restriction on application type. The two propositions KVM and Xen are quite equivalent, we have chosen the first one because of previous work experience with it.

As we propose to gather two different physical architectures, ARM and x86, in the same infrastructure, it means we also have to choose between two alternatives for the virtual machines architecture. When the virtual and the physical machines share the same architecture then we benefit from the virtualization extensions. On the contrary, if the two architectures are different, we have to use emulation. Emulation is a concept which allows to execute programs compiled for an architecture different from the host one. It consists in an hardware abstraction and the program will be executed through dynamic translation of the binary instructions.

For this purpose we have chosen QEMU emulator because it is closely related to KVM. In fact QEMU can detect if the virtual machine and the host have the same architecture, in this case emulation is not needed and it automatically uses virtualization extensions of the hardware. Hence our idea is based on the assumption that it could be possible to migrate one virtual machine of fixed architecture between two different hosts. During the migration, the system should just have to switch from emulation to virtualization extensions, or the opposite, according to the architecture of the source and

destination hosts. Status of our work about migration between heterogeneous architectures is detailed in section IV-B.

Figure 4 pictures the two alternatives for the virtual machine and their underlying functioning. First and last cases have low overhead thanks to the virtualization extensions while the two cases needing software translation suffer from a high performance impact. The resulting overhead of emulation is discussed in section IV-A. Although emulation adds an important overhead, we still assume that low power ARM processors will bring more energy efficiency, especially for low load, because their static costs are much smaller than those of regular x86 servers.



(a) x86 VM (b) ARM VM

Fig. 4. Two alternatives for VM architecture and their underlying layers : Emulation or Virtualization extensions

C. Experimental hardware

TABLE II gathers the hardware selected for our experiments, with their detailed characteristics. We have chosen the ARM Cortex-A15 processor for its low power, its good performances and its virtualization extensions. It is a quite recent processor, its first implementation was done by Samsung with the Exynos5250 SoC. The first device powered by this chip is the Samsung Chromebook released in 2012. As the code name suggests, this notebook comes with Google's Chrome Operating System, but to be able to use KVM software and virtualization extensions we need a Linux distribution. Moreover as mentioned in Table I, the Linux kernel version should be equal or posterior to 3.9. We have managed to make the Samsung Chromebook boot an Ubuntu 12.04 with a Linux kernel 3.13, and installed QEMU 2.0 and KVM for virtualization extensions. Concerning the experiments with binaries execution which do not require virtualization, we use an Ubuntu distribution based on the ChromeOS kernel already installed. In this case to get power consumption information we use powerstat Ubuntu package that gets monitoring data from the battery via ACPI.

For x86 architecture the choice is much larger. In order to benefit from servers with power monitoring, we have run our experiments on servers from the Grid'5000 testbed [17]. Grid'5000 is a French experimental platform, geographically distributed over 11 sites in France and Luxembourg, dedicated to scientific research concerning large scale infrastructures. We have chosen a server with an Intel Xeon processor and another with an AMD Opteron processor located respectively in monitored clusters of Lyon and 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 operating system with QEMU 1.7 installed. In Lyon, electrical consumption is acquired thanks to watt-meters from Omegawatt and accessible on Grid'5000 intranet, whereas in Rennes monitored PDU from EATON are used and power data is fetched via SNMP requests.

TABLE II. SUMMARY OF SELECTED HARDWARE

Fullname	Samsung Chromebook	Dell PowerEdge R720	HP Proliant DL165 G7	HP 7800 Workstation
Architecture	ARMv7 32 bits	x86 Intel 64 bits	x86 AMD 64 bits	x86 Intel 64 bits
CPU	2 x ARM Cortex-A15	2 x Intel Xeon E5-2630	2 x AMD Opteron 6164	2 x Intel Xeon E5620
Total cores	2	12	24	8
Power consumption	5 – 25 W	96 – 227 W	180 – 280 W	149 - 248 W
Release year	2012	2012	2010	2013

An important observation we can make here is the huge difference between idle consumptions. The idle power of the HP Proliant for instance is more than 20 times greater than the one of the Samsung Chromebook, and 2 times greater than the one of the Dell PowerEdge. The upper power bound corresponds to the maximum measured power consumption when all cores of the processor are fully loaded. Of course the energy consumption is not the only noteworthy difference between these machines, performance is the other aspect to consider, and is discussed in following sections.

IV. INFRASTRUCTURE VALIDATION AND CALIBRATION MEASUREMENTS

Through this section we want to validate the feasibility of our proposed infrastructure and expose the methodology of profiling introduced in Fig.3 thanks to real experiments. We evaluate the respective performance of each hardware and characterize the cost of emulation. These experiments and profiles are created for a specific application but of course the same process can be done with other kinds of targeted applications. Afterwards, we investigate how migrations of virtual machine between heterogeneous hosts is possible and what are the associated impacts.

A. Emulation overhead

As mentioned earlier, we suggest to use emulation, but this piece of software performing binary translations implies an overhead both in performance and energy consumption. We need to measure this impact to evaluate is emulation is a good perspective. As x86 virtual machine on ARM host is not fully functional at the time we are writing, we made experiments with QEMU User Emulation which allows to execute binaries compiled for a different architecture by dynamically translating the instructions during the execution. Not all programs can be executed with this type of dynamic translation, we need an application compiled with statically linked libraries. For this purpose, we have chosen the nbench [18] benchmark program. It is a simple application written in C, which is composed of several subprograms designed to test CPU capabilities of a machine.

TABLE III shows the overhead of emulation, by dynamic translation, for each selected hardware and for two types of computing benchmark. Column 'Int' refers to Idea encryption

program, and 'Float' to Fourier algorithm of nbench benchmark. The first column is the maximum number of iterations per second for a native execution, and the second one is for an execution of the 'opposite' architecture binary via QEMU user emulation program.

TABLE III. OVERHEAD OF EMULATION FOR EACH HARDWARE EXECUTION OF IDEA BENCHMARK

	Native		Emulation		Overhead	
Processor name	Int	Float	Int	Float	Int	Float
ARM Cortex-A15 (Little)	8233,9	27251	932,5	604,1	8,8	45,1
x86 Intel Xeon (Medium)	102893,9	380437	11479,2	11153,1	8,9	34,1
x86 AMD Opteron (Big)	113569,8	320823	15239,5	12599,8	7,4	25,5

The last column whose title is 'Overhead' represents the ratio between emulation performances 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 around 7 to 9 times slower, while for float computation the overhead is much important, from 25 to 45 times, the largest difference being for the ARM processor. Even if x86 processors are natively more powerful than ARM ones, (about 12 to 13 times in our examples) the overhead causes the emulation to slow down 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 an ARM compiled float computation, the ARM Cortex-A15 reaches 27251 iterations per second whereas x86 Intel Xeon and AMD Opteron reach respectively 11153,03 and 12599,76 iterations per second. Therefore, the choice of target architecture for the executed program is very important and must be suited to the application type.

In order to stick to the ARM big.LITTLE spirit, introduced in section II, and to ease the designation of the processor, we adopt code names. In this manner, ARM Chromebook is the "Little", Intel Xeon from Grid'5000 Lyon is the "Medium" and AMD Opteron from Grid'5000 Rennes is the "Big". We attribute these names to processors according to their maximum performance for this chosen application, here expressed in number of iterations per second. Thereby, these processors code names are not definitive and are only valid for this selected application. As those processors offer different hardware characteristics, their behaviors and performances are not the same when running different applications.

The aim of the two following figures (Fig.5 and 6) is to compare the two solutions for the virtual machine architecture as depicted in Fig.4. Except here, as mentioned earlier, we are not dealing with full virtual machines but only with binaries execution, natively or through dynamic translation. Fig.5 and 6 show the average power consumption for an evolving number of iterations per second, from 0 to maximum, of the Idea encryption benchmark from nbench. The curve starting point is the average power consumption at idle state, and the ending point of each curve corresponds to the average power during a complete execution of the benchmark. We have slightly modified the nbench benchmark by introducing 'nanosleep' calls in order to reduce the maximum performance and then get more data points. We have chosen to run the benchmark five times with five different durations of sleep for which we get the maximum number of iterations per second reached and the average power consumption during the execution. We have in total 5 data points for each hardware curve and we approach these points with a linear fitting.

Each graph plots three curves corresponding to our three selected hardware presented in TABLE II. The most powerful is "Big" the server from Grid'5000 Rennes cluster, and it defines the maximum scale of our graphic. The two left curves are endless because we reproduce the power consumption scheme we obtain for one machine as if we can have several machines of each type and cumulate their performance. The least powerful hardware is the Chromebook, the "Little" processor of our platform, plotted in green, 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 "Little" processor is symbolized by the vertical purple dashed line. On the opposite, when we repeat the "Medium" from Grid'5000 Lyon, it shortly becomes out of scale because its static idle consumption is too important.

Fig.5 corresponds to the case depicted in Fig.4(b) where the executed program is compiled for ARM architecture. The program is executed natively on the "Little" processor, green curve, and through dynamic translation on the "Medium" and "Big" processors, blue and red curves. On the opposite, Fig.6 represents what happens in the case of Fig.4(a) where the target architecture is x86 and the emulation only concerns the "Little" processor. 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 introduced in section IV-A. The overall total performance is reduced by 7.45 times when we use an ARM binary.

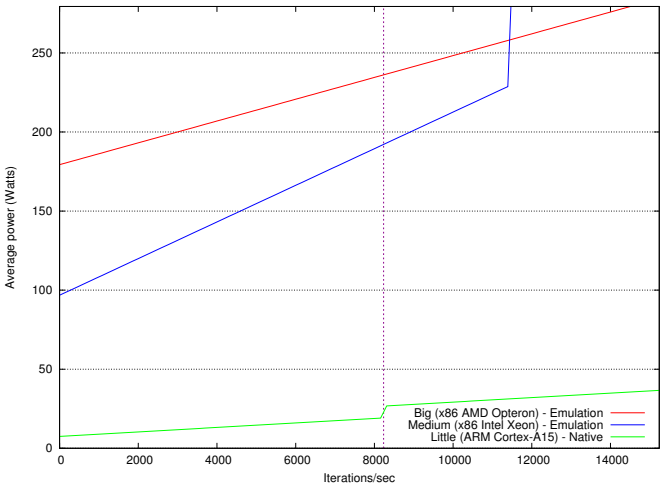


Fig. 5. Average power consumption (watts) according to number of iterations per second of the same ARM program (IDEA benchmark) on 3 different types of hardware

For the ARM program on Fig. 5, we see that x86 architectures perform quite poorly, and if the execution of the program could be parallelized on two ARM platforms, then it would always be the most relevant configuration concerning energy consumption. If we cannot consider parallelization of the program, then the "Medium" machine would be the chosen host from approximately 8000 to 110000 iterations per second,

and the "Big" one would be elected passed this threshold of performance. As it can be seen, ARM hardware leads to huge energy savings, in fact the green very low 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. 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 bring even more performances.

On the other hand, for x86 program on Fig.6, the performance of ARM platform is very low because it is reduced due to dynamic translation. Consequently, we can observe on the zoom area that the "Little" processor 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 possible. Considering the last perspective, the gains from ARM hardware are only profitable for a reduced part of low performance (the first 1/30th of the total performance), that we can only see on the zoomed part of the graph. The most predominant hardware is the "Medium" one, we realize that the "Big" machine only brings a small improvement in performance but consumes a lot more than the "Medium" most of the time. This can be justified by the fact that the Dell PowerEdge R720 ("Medium" machine) is the most recent server of our selection, and the energy efficiency aspect must have been better considered during its design.

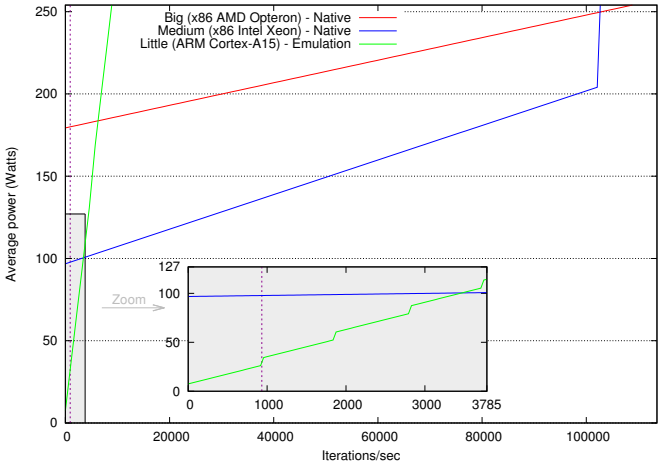


Fig. 6. Average power consumption (watts) according to number of iterations per second of the same x86 program (IDEA benchmark) on 3 different types of hardware

From the observations we just made, we can say that the choice of the architecture must be dependent from the applications type and profile. Moreover, it would be interesting to find other pieces of hardware that would fit between the "Little" and the "Medium" machines. The ARM Cortex-A15 is a great low power processor that brings promising energy savings, but the performance gap between itself and x86 servers is too large.

B. *Impacts of live migration*

We have performed some experiments of live migration with an ARM based virtual machine. For this purpose we used Libvirt version 1.2.9 as VM manager. Hardware used is an HP 7800 Workstation with an Intel Xeon E5620 CPU, and the previously introduced Samsung Chromebook. They are both monitored with external watt-meters Watts'upPro and power data is acquired and stored via Kwapi API [19]. At the current status of our experiments, only migration from the HP Workstation the Chromebook works. Figure 7 presents the extra power consumption of each host during the process of virtual machine migration. In fact in order to focus only on the overhead consumption implied by the migration, we have removed the static idle consumption.

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, as it can be seen on Fig.7 at time t = 5 seconds, the power consumption increases slightly. This corresponds to the moment when the server starts receiving the virtual machine.

These are some first steps in our work about heterogeneous migrations between Big and Little. We will continue our investigations about all the parameters which can affect the migration behavior and see how they can be enhanced.

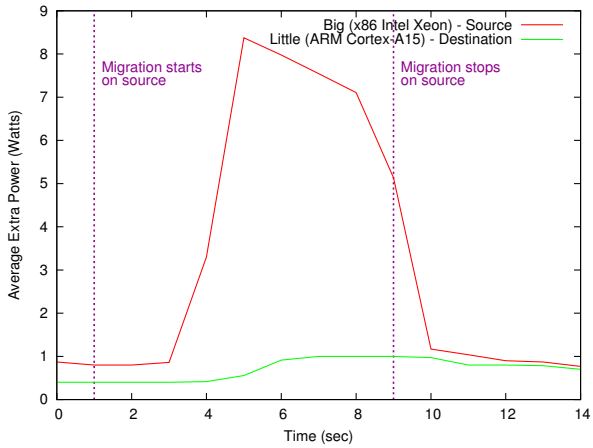


Fig. 7. Extra power consumption (Watts) during live migration of ARM virtual machine from Big (HP 7800 Workstation) to Little (Samsung Chromebook)

Through this section we have shown the feasibility of our proposed infrastructure to generalize the concept of big.LITTLE architecture at datacenter scale. Emulation and live migration allow us to move virtual machines across heterogeneous architectures. The major drawback of this solution is the quite important overhead implied by emulation. This comes from our first wishes to build a solution which applies to any type of applications. Yet it is possible to provide solutions with better performances by studying and classifying applications according to their characteristics.

The two main categories are stateless and statefull applications. Of course the last type is the hardest to treat because it means that during a migration the state should be carried

with the application. The transfer of the state then becomes the crucial part because there can be some restrictions on it depending on the application. For example the state may require some modifications in order to be readable by an other architecture. So the duration of the migration will be increased, and will also stretch out with the size of the state. In some other cases maybe the migration will only be possible at certain predefined checkpoints. And probably some statefull applications will not support any kind of migrations. These more complex types require detailed studies to find better suited solutions. For the rest of this article we are considering the case of stateless web servers. For this kind of application, migration is easier because it consists in shutdown an instance and restart another instance elsewhere. We do not necessarily need emulation which simplifies the problem, but there are other parameters to manage such as the distribution of requests among servers, especially when they are migrating.

V. BML SCHEDULING FRAMEWORK

Fig.8 shows the different bricks of our scheduling framework. The two major starting points are, on one side the heterogeneous computing resources described in section **??**, and on the other side the applications to run. The results expected from the framework are the decisions concerning where to execute the applications as well as an intelligent management of the resources. The key leverage to benefit from heterogeneous resources is the migration of applications between the physical machines according to their needs. Besides, the resource management consists in controlling the state of the machines. When they are unused we want them to consume as less as possible, meaning to be switched off or put in a low power mode. But when the resources are needed, they should be available without delay. Actions of switching off and on are not trivial because they take time and energy. Hence, resource management policies have to be developed and tested with care to evaluate the quality of the decisions regarding energy but also availability of resources.

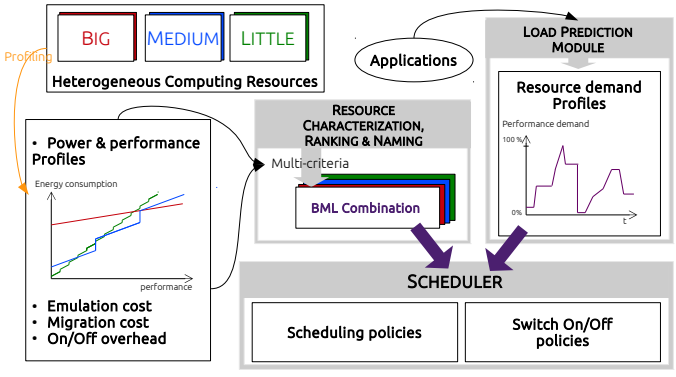


Fig. 8. Scheduling framework architecture

As scheduling policies are concerned, to be able to make efficient decisions, they must rely on detailed and concrete information. These information is represented by the two profiles parts on Fig.8. On the left, hardware power and performance profiles are derived from performance benchmarking and power profiling discussed on Fig.3. For each level of performance this profile gives the corresponding energy consumption for each resource type. All these profiles, together with other parameters such as overhead costs for emulation, migration and On/Off, are then used to characterize, rank, and name our computing resources. The result of this analysis is the BML combination, standing for Big,Medium,Little. It consists in finding the best combination of resources, meaning the least consuming, to execute an application depending on its performance needs. Later in the section VI-C we detail how to create this combination of hardware for a specific application use case.

The graph on the right represents a resource demand profile from an application. This profile describes the evolution of resource needs from an application over time. Such information is necessary to find the most suitable machine to execute the application at each moment. The challenge is how to get such profiles. Of course this is closely dependent to the type of applications. The easiest case is when it is possible to perfectly know the application profile before running it. Further in this paper this is the case we assume for our simulations. In a more generic situation, when the applications are not known in advance, then the resource demand profiles are more difficult to build, and a system of predictions is required. The prediction module will monitor the execution of the application by analyzing how much resource it uses. Based on these observations, a mechanism will predict the resources usages for the future.

The heart of our framework is the scheduling policies. The profiles just described represent the inputs of the decisions made by the scheduler. All along the execution of the applications, based on future resource needs, the scheduler is choosing the location of execution, making sure the performance requirements are fulfilled and that no extra energy is wasted. Our scheduling policies consist in associating a certain level of performance to a specific combination of hardware. In the rest of the paper, we will focus mainly on the scheduler module, with explanations about the characterization, ranking and naming phase. The prediction module, which represents an area of research in itself, is not yet implemented and needs to be explored as future work.

VI. BML SCHEDULER VALIDATION

A. Simulation environment

In this section we detail our simulation environment for heterogeneous computing. We choose to validate with a focus on stateless applications. After the proof of concept from section III, we want to see what can be the gains of our solution at a larger scale. Simulation offer the possibility to evaluate our ideas in an easier, and also quicker, way than experimentation. Our simulations are of course based on measurements acquired during real experiments. The main advantage is to be able to test and validate our solution on several types of application, as well as on any kind of datacenter by modifying the inputs of the simulator. We want to consider a datacenter gathering heterogeneous machines, and be able to make the best use of them to approach energy proportionality. Of course, the most energy efficient the servers are, the most interesting the results will be. In order to exploit the heterogeneity of the datacenter and have the most efficient utilization, scheduling decisions have to be taken carefully. Thus we have developed a solution

which considers all characteristics of available servers together with resource needs to find the most suitable configuration.

Technically speaking, we have developed our own simulator in Python language. Inputs mainly consist of architectures profiles for the targeted application. Acquisition of these profiles is fully described in part VI-B. As output, our simulator provides the energy consumption for a chosen scenario, but also the detailed state of the datacenter over time. Meaning we want to know how many machines are turned on and when, in order to clearly see where does the energy consumption come from, and how decisions are taken during our simulations. These results will show where improvements can be done and then allow us to develop even more efficient scheduling policies.

B. *Profiles acquisition methodology*

Profiles acquisition is the key step to obtain reliable and realistic inputs for our simulator. We have studied in details each type of selected hardware to evaluate the potential benefits of our heterogeneous platform as a whole. The set of machines is the same as described in TABLE II. We have deployed the same environment on all machines and run the experiments with the same conditions, for a specific use case. This allows us to obtain comparable data and thus gives us precise information on each hardware behavior. The use case we have chosen is a web server. This type of service offers an important variability in terms of demand, which implies that some efforts have to be done to better suit to demand evolution. Indeed, the challenge in web server provisioning is to be able to answer all requests with an acceptable latency, but without over-provisioning because we want to save as much energy as possible.

We describe our experimental methodology for creating profiles of web server application for each type of hardware. These profiles are built around a set of measures which concern performance, quality of service and energy consumption. Here are our chosen experimental configurations : We use lighttpd as web server. The requested web page is a python cgi script returning randomly an image among five different images, whose size is between 1 and 3 KB. To generate the requests we use Siege which is an http load testing and benchmarking tool. Beside generating clients and requests to the given web page, Siege retrieves data about the number of successfully answered requests, the response time, the latency, the amount of data transferred and so on. Our objective is to get the maximum number of requests answered in one second for each type of server, as well as the power consumption associated with it. To do so, we load the web server with an increasing number of concurrent clients in order to make the number of requests per second increase too. At one point, the number of requests per second stagnates, and the latency starts increasing. By computing the average number of requests per second for a latency greater than a given duration, we get what we consider the maximum number of requests answered for an architecture, with a guaranteed quality of service.

Fig. 9 pictures this result for the Intel Xeon processor. The vertical black line represents the point when the maximum number of requests is reached. The green curve is the power profile. During the execution of Siege benchmark the power of the node hosting the web server is monitored, one value per second, and we compute the average power consumption for each number of concurrent clients. For this architecture, the rate of successfully requests stabilizes just under 900 requests per second. We can clearly see that since this value is reached, the latency is continuously increasing. The power consumption, after rising from approximately to 175 Watts, is also stable.

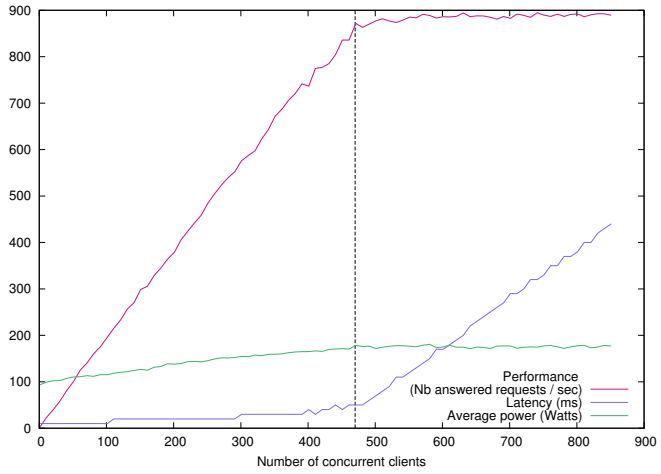


Fig. 9. Performance, quality of service and power profiles of lighttpd web server running on x86 Intel Xeon (Big) requested with different number of concurrent clients with Siege

We have repeated this technique for each server type, and gathered in TABLE IV the simplified profiles, consisting in three values : average maximum number of requests treated per second, average power consumption during this period, and average idle power consumption. Machines are ordered in this table by descending performance rate. Following our naming convention previously explained concerning TABLE III, each processor gets a code name which is decided by its maximum performance. For this specific application, the least powerful processor is the ARM Cortex-A15, but also the least consuming by far. We will then refer to this processor as the 'Little'. Surprisingly, there is a difference of performance between the two other machines compared to the results in section III. The machine with the Intel Xeon processor is more powerful than the one with the AMD Opteron, but also consuming a lot less. This difference is due to the fact that we are only profiling one web server which only runs on one core of each machine. This way we do not see like in section III where all cores where busy with nbench benchmark that the AMD Opteron is more powerful thanks to its greater number of cores. These results are also explained by the fact that these two machines are from different generations with 6 years difference. As a result for this use case the 'Big' processor is the Intel Xeon whereas the 'Medium' one is the AMD Opteron.

TABLE IV. WEB SERVER PERFORMANCE AND POWER PROFILES FOR EACH ARCHITECTURE

Processor name	Max performance rate (nbReqs/sec)	Max Average Power (W)	Idle Power (W)
x86 Intel Xeon (Big)	888,74	175,25	93.61
x86 AMD Opteron (Medium)	583,35	221,16	172
ARM Cortex-A15 (Little)	31,54	11,96	5,5

Once these profiles are acquired, we can derive them to deduce the needed information for our simulations. Indeed it is necessary to be able to get the power consumption for a specific level of demand. This analysis is crucial because we will rely on it to take scheduling decisions to minimize the energy consumption of our heterogeneous platform.

Therefore, we interpolate the profiles just acquired and repeated them as if we can have as many servers as we would need. These graphical profiles are gathered in Fig. 10. From this graphic we can picture better the difference between different architectures in terms of performance and energy consumption. More importantly, it allows to clearly see which architecture is the least consuming for each level of performance. In fact we can notice for example that until a load of about 350 requests per second, 11 'Little' nodes are just equivalent to 1 'Big' concerning power consumption. The most important conclusion we can draw with this graph is that with these selected processors, the ARM Cortex-A15 and the Intel Xeon are the most energy efficient ones. For the rest of the paper we will not take into account the AMD Opteron processor because it does not offer interesting characteristics for this selected application. Graphically the curve plotted in blue is always above the two others which means that the 'Medium' processor would never be an good option. In next part, we detail how we compose an heterogeneous combination, that we name "BML combination", out of these two architectures 'Little' and 'Big' and consequently how we reduce the energy consumption.

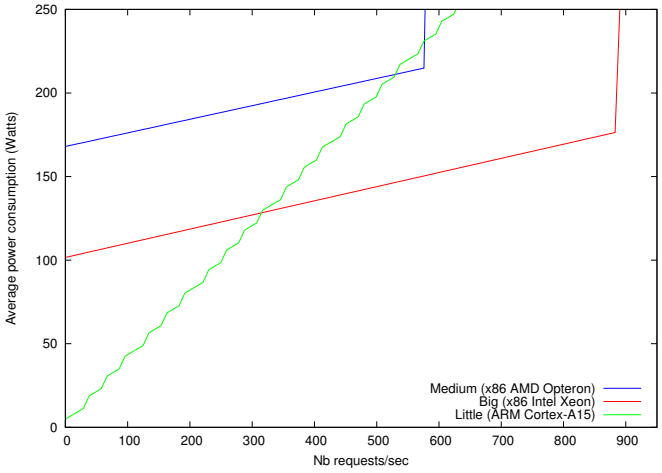


Fig. 10. Performance and power profiles of lighttpd web server running on three different architectures

C. Creation of Big-Medium-Little (BML) combination

Performance and energy profiles just made from experiments can now be analyzed to find combinations of different architectures that can bring more interesting results than choosing only one type of server at a time. On Fig. 11, the dashed curve is the BML combination profile, mixing the two most energy efficient nodes of our platform that we have name 'Big' and 'Little'. From this graph, we can see that when the web server load is less than 340 request per second, using several 'Little' nodes is the best solution. Then from this point, one 'Big' node is more efficient, until its maximum number of requests it can answer. When the demand is just a little higher than maximum 'Big' performance, it is worthless turning on another 'Big' node, but it is wiser to turn on 'Little' ones instead. Several results with associated gains for this combination are shown later.

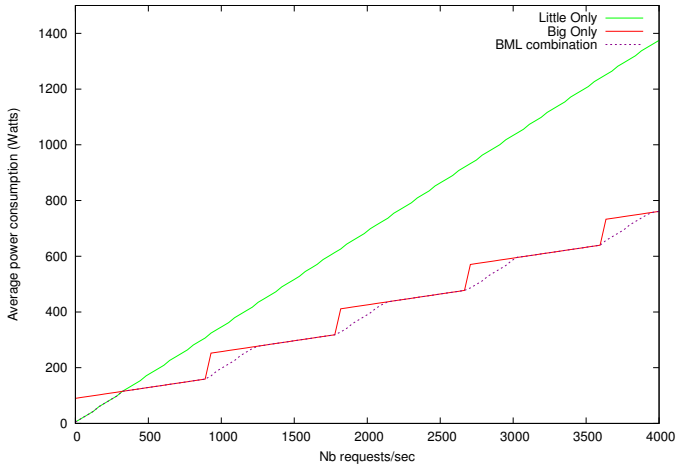


Fig. 11. Creation of the BML combination profile for lighttpd web server, which is the best energy proportional profile with our selected hardware

In order to find this BML combination profile, the first step is to compute the junction point between the two architectures. This specific point is expressed in number of Requests per second. It corresponds to the point when several 'Little' nodes become more consuming than one 'Big' node. To compute this point technically, it consists in reading simultaneously the two profiles and find the threshold rate from when the power consumption of the architecture considered as 'Little' is greater than the one of the 'Big' architecture. We refer as 'J' to this threshold point.

Once the threshold J has been found, here is the algorithm to compute the power consumption of the BML combination for a given requests rate. The first step (lines 1-2) is to compute how many 'Big' nodes can be fully loaded, meaning giving their maximum performance. Indeed we have previously seen that servers are the most efficient at their maximum load. Next step (lines 3-6) is to find which architecture will answer the remaining requests. This is where the threshold J previously computed is needed. If the remaining requests number is less than J then 'Little' nodes are chosen whereas if it is greater than J one 'Big' node is preferred. When the architecture for remaining requests is found, the last step is to gather both energy consumptions in one global value.

D. Gains of BML combination for different scenarios

We have chosen to work with web servers because of the important load variability. Moreover, as we are running simulations, we can study different use cases by modifying the input requests traces given to our simulator. In this paper we are presenting simulations done with the log traces of the 1998 World Cup website (available at http://ita.ee.lbl.gov/html/contrib/WorldCup.html). Traces have been collected during a period of 4 months, between April and July a 1998. Total received number of requests is just

Algorithm 1 Compute BML combination and its $Power$ to answer $nbReqs$ requests

$nbBig \leftarrow getNbNodesFor('Big', nbReqs)$
$fullBigPower \leftarrow nbBig * bigMaxPower$
$remainingReqs \leftarrow nbReqs - (nbBig * bigMaxPerf)$
if $remainingReqs < J$ **then**
  $arch \leftarrow'' Little''$
else
  $arch \leftarrow'' Big''$
end if
$TotalPower \leftarrow fullBigPower + getPowerFor(arch, remainingReqs)$

over one billion. On Fig. 12 is plotted the distribution of the requests over time. In red is the number of requests for each second and in blue is the mean number of requests per second for each day. According to the mean value, we can deduce that the variation in demand is very large during one day. 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 only of 900 requests per second.

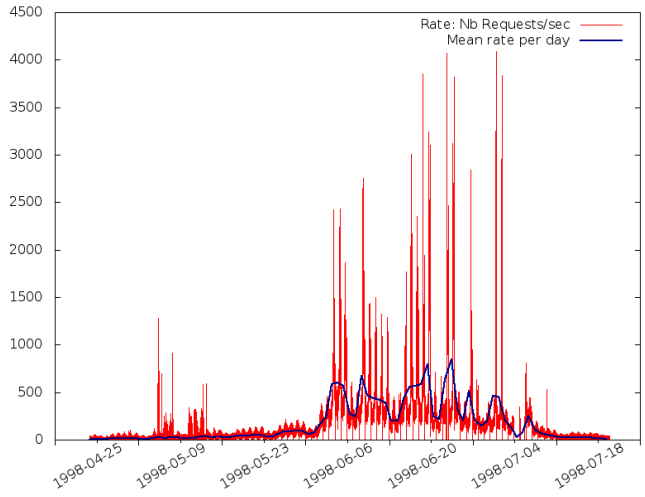


Fig. 12. High variability of traces log distribution of 1998 WorldCup website (one value per second compared to mean rate of the day)

In a first step, we have simulated the execution of the web servers as if we could have an homogeneous datacenter of each machine type, with an infinite number of nodes. We have also done the simulations for the BML combination, which is the only heterogeneous computing on the graph. The results are plotted in Fig. 13. Each curve represents what would be the energy consumption per day if the web servers for the 1998 World Cup were hosted on a datacenter composed of this type of machines. The powering on and off of machines is not taking into account here. At each time unit we only take into account the power consumption of the needed number of machines to answer all the requests. Moreover, our simulations work with the complete requests traces with an entire knowledge of the future arrivals of requests. Our framework does not contains yet the "Prediction module" as depicted in Fig. 8.

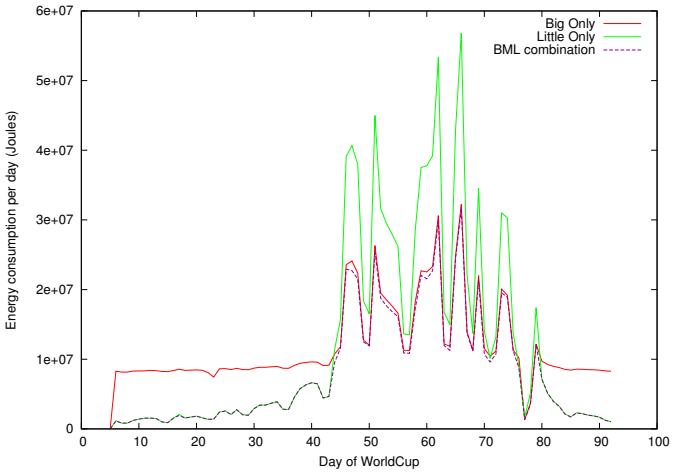


Fig. 13. Energy consumption cumulated per day (in Joules) for three different composition of datacenter : 'Big' only, 'Little' only and BML combination

On the graph, we can find the same pattern given earlier by the architectures profiling, meaning that, when the rate is quite low, then 'Little only' datacenter is the least consuming, but when the load is higher, during the actual period of the World Cup, then the 'Big only' option is more interesting. It has to be noted that on this graph only the cumulated energy consumption per day is represented, so the high variations of requests during night and day are hidden.

Because we realized that our results depends mostly on the absolute requests rate, we have run our simulations with different alterations of the base logs described in Fig.12. All the results are presented in Fig. 14. We have computed the total energy consumption for each solution for the whole World Cup scenario. We have only selected the two most interesting solutions : 'Big' and 'Little' as well as the BML combination. Except the BML combination solution that is always the least consuming, the second one is not always the same, it depends on the overall number of requests. To highlight this we have computed the percentage of gains of the BML solution over the two others. On Fig. 14, the written percentage is always the gains of BML combination over the least consuming solution between 'Big Only' or 'Little only'. In fact, it is only for the baseline traces that 'Little only' is more interesting than the other alternative. For all greater multiplier than one, the percentages always represent the gains of BML combination over the 'Big only' solution. The more important is the number of total requests and the least are the gains of BML solution because it is mainly composed of 'Big' nodes, more suited to answer high rate.

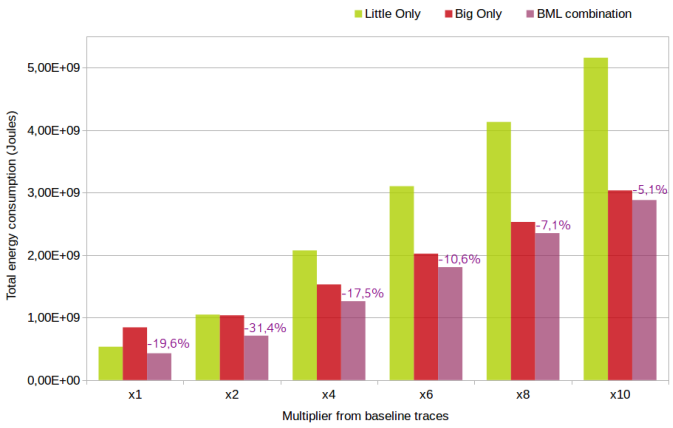


Fig. 14. Total energy consumption for variants of the baseline World Cup traces and Energy gains of BML combination over homogeneous solutions

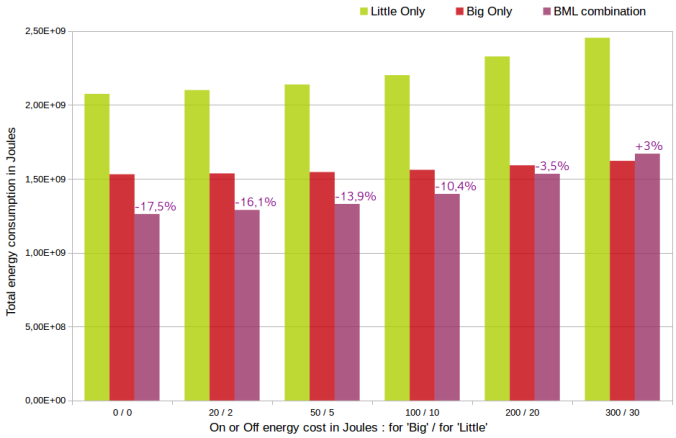


Fig. 15. Gains of BML combination for baseline traces multiplied by 4, for different On/Off energy costs for 'Big' and 'Little' nodes (in Joules)

This Fig.14 pictures the best cases and maximum gains we can get from the BML combination. In fact, until now we do not take into account any overhead for powering On and Off the nodes, nor for the web server migrations. Therefore, to get closer to reality we decide to tackle On/Off overheads, and build the module of our framework for On/Off policies.

Firstly we just included the overhead to the already presented results of Fig.14, where the state of the machines is updated every time unit, at the same time at the rate is evolving. Concerning energy, the cost of each On/Off action should not be too high to, otherwise the gains of the BML combination would be lost by the costly On/Off decisions. This is what is shown with Fig.15. It represents the total energy consumption for the baseline traces multiplied by four, which is approximately our average case. We specify different energy cost values for On/On actions to see how results are evolving. For the sake of simplicity, in our simulations we assume that switching on or switching off a node consume the same amount of energy. As 'Big' and 'Little' approximately have a factor ten difference in energy consumption, we assume a factor ten difference for On/Off costs. For example on the Fig.15 the '100/10' case means that we consider an energy overhead of 100 Joules for each action of powering off or on a 'Big' node, and 10 Joules for 'Little' node. For each set of energy costs is displayed the total energy for three scenerios : in green web servers are hosted in a datacenter composed of only 'Little' nodes, in red is only 'Big' nodes, while in purple it is the BML combination of 'Big' and 'Little' nodes. For this scenario if we consider only homogeneous solutions, the 'Big' option is always the less consuming one. Indeed, we compute the percentage of gains of the BML combination over the most interesting solution which is 'Big' only. We write this percentage in purple to see more clearly the difference between these two cases. For the highest On/Off overheads, the energy consumption of the BML combination becomes greater than the homogeneous 'Big' cluster. Considering the obtained results, we could conclude that until '200/20' On/Off overhead it is interesting to consider the BML combination. This shows that it is crucial to take into account On/Off overheads to consider our proposition. Moreover, this also shows that more intelligent On/Off policies must be developed to increase the gains.

VII. Conclusions and Perspectives

In this article we propose and study a solution to bring the concept from "ARM big.LITTLE" to datacenters and extend it to "Big-Medium-Little" to reach some energy proportionality for HPC and clouds infrastructures. We propose an infrastructure composed of heterogeneous computing resources, as well as a framework for applications scheduling and resource managements. We address some technical issues to deal with heterogeneous architectures, and run simulations to validate our proposition for a stateless web server use case. We show that distributing requests among a combination of heterogeneous nodes bring energy savings compared to an homogeneous datacenter. To get closer to real implementation we tackle the issue of overhead for powering On and Off the nodes. In this sense, some work still need to be done. We want to implement more efficient resource management policies by taking as less On/Off decisions as possible while still minimizing energy consumption. Another step would be to specify the topology of the datacenter as input of the simulator in order to make scheduling decisions only with available resources. With this last perspective, the quality of service will be a new parameter to take into account. Lastly to have a complete framework, the prediction module should be developed to be able to run any applications and predict its evolution from monitoring its past behavior.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://grid5000.fr). This work is supported by the Inria Hemera Large Scale Initiative. This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] W.V. Heddeghem, S. Lambert, B. Lannoo, D. Colle, M. Pickavet, and P. Demeester. Trends in Worldwide ICT Electricity Consumption from 2007 to 2012. *Computer Communications*, 50, 2014.

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

[3] L.A. Barroso and U. Holzle. The Case for Energy-Proportional Computing. *IEEE Computer*, 2007.

[4] 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.

[5] 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.

[6] 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.

[7] G.L. Tsafack Chetsa, L. Lefèvre, J-M. Pierson, P. Stolf, and G. Da Costa. Exploiting Performance Counters to Predict and Improve Energy Performance of HPC Systems. *Future Generation Computer Systems*, 2014.

[8] 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.

[9] F. Ryckbosch, S. Polfliet, and L. Eeckhout. Trends in server energy proportionality. *Computer*, 2011.

[10] 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*, June 2014.

[11] ARM big.LITTLE Technology: The Future of Mobile. *ARM White Paper*, 2013.

[12] NVIDIA Tegra K1 : A New Era in Mobile Computing. *NVIDIA White Paper*, 2014.

[13] G. Da Costa. Heterogeneity: The Key to Achieve Power-Proportional Computing. *IEEE International Symposium on Cluster Computing and the Grid*, 2013.

[14] V. Villebonnet, G. Da Costa, L. Lefevre, J-M. Pierson, and P. Stolf. Towards Generalizing "Big.Little" for Energy Proportional HPC and Cloud Infrastructures. In *IEEE International Conference on Sustainable Computing and Communications (SustainCom 2014)*, Sydney, Australia, December 2014.

[15] HP. HP Project Moonshot and the Redstone Development Server Platform. http://h10032.www1.hp.com/ctg/Manual/c03442116.pdf, May 2013.

[16] CRIU. Checkpoint/Restore In Userspace. http://www.criu.org/.

[17] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, Fl. Quesnel, C. Rohr, and L. Sarzyniec. Adding virtualization capabilities to the Grid'5000 testbed. In *Cloud Computing and Services Science*, Communications in Computer and Information Science. Springer International Publishing, 2013.

[18] U.F. Mayer. Linux/Unix Nbench. http://www.tux.org/~mayer/linux/bmark.html.

[19] 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 *The 4th IEEE International Conference on Sustainable Computing and Communications*, December 2014.