DVFS governor for HPC: Higher, Faster, Greener

Georges da Costa, Jean-Marc Pierson

DVFS governor for HPC: Higher, Faster, Greener

Georges Da Costa, Jean-Marc Pierson
IRIT, University of Toulouse
dacosta@irit.fr pierson@irit.fr

***Abstract*—In High Performance Computing, being respectful of the environment is usually secondary compared to performance: The faster, the better. As Exascale computing is in the spotlight, electric power concerns arise as current exascale projects might need too much power to even boot. A recent incentive (*Exascale at maximum 20MW*) shows that reality is catching up with HPC center designers.**

Beyond classical works on hardware infrastructure or at the middleware level, we do believe that system-level solutions have great potential for energy reduction. Moreover energy-reduction has often been neglected by the HPC community that focus mainly on raw computing performance.

In the literature, energy savings is achieved mainly by two means: Either processor load is the only metric taken into account to reduce processors frequency and to ensure no impact on raw performances; Or processor frequency is managed only at task level outside the critical path.

In this article we show that designing and implementing a DVFS (Dynamic Voltage and Frequency Scaling) mechanism based on instantaneous system values (here network activity) can save up to 25% of energy consumption while reducing marginally performance. In several cases, reducing energy consumption also leads to an increase in performances because of the thermal budget of recent processors. This work is validated with real experiments on a linux cluster using the NAS Parallel Benchmark (NPB).

***Index Terms*—HPC; Green; Energy; DVFS; Governor;**

I. Introduction

Saving energy for High Performance Computing applications has gained interest since only a few years. Solutions widely accepted in other IT domains can not be directly transferred to HPC systems. In HPC, the key performance criteria is time-to-solution. Consolidation with virtualization and dynamic migration of virtual machines coupled with switching-off unused nodes, empowered in commodity clusters and cloud computing, are not acceptable because of their impact on performances and the high level of utilization of the HPC clusters. Applications are deployed on large scale infrastructure making solutions for embedded systems also not applicable.

Techniques for reducing the energy demand on HPC infrastructure are mainly based on hardware dynamic adaptation to the applications needs. For instance when an application is using the disk IO only for specific phases (usually initialization and loading phase, and finish phase and dumping results), disks can be switched into an energy efficient sleep mode for the rest of the time. The same applies for communication phases when the CPU can be slowed down without impacting the application makespan. Diverse techniques exist for detecting such possibilities, either before, during or after the execution of an HPC application.

In this article, we show the potential for energy savings using automatic adaptation to application behavior at runtime. We propose a simple algorithm (NetSched) for optimizing HPC energy-to-solution based on CPU frequency adaptation on the hosts of the infrastructure, and we validate our approach using well known HPC benchmarks. Our main contributions are the following:

- Development of system-level runtime monitoring of HPC applications enabling behavior detection for HPC applications;
- Development of an algorithm for automatic hardware adjustment aimed at reducing energy consumption without impacting time-to-solution.

The rest of this article is organized as follows: Section II gives an overview of the different approaches for energy savings in the context of HPC applications. Section III details our methodology and our NetSched algorithm while Section IV describes our experiment setup and results. Section V discusses the impact of our results for HPC centers while Section VI concludes and gives perspectives to our work.

II. State of the Art

In HPC environments the main focus for reducing energy has long been to improve the power efficiency of the hardware [7], including the environment infrastructure (i.e. cooling efficiency). While incomplete, [19] gives a good overview of the literature on power management for high performance systems.

Another widely used solution for cloud computing is consolidation. But even with low-overhead virtualization techniques for HPC systems [16], [26], [28] consolidation is of no use on fully loaded systems.

In the literature, the method usually followed to reduce energy in HPC systems is to consider that HPC workload can be modeled as a graph of tasks. For tasks not on the critical path in terms of execution time, it is possible to reduce their processors' frequency. It leads to energy-savings and reduced performance for those tasks. However as they are not on the critical path, the makespan of the whole application is not affected. This technique used in Adagio [24] is more generally called slack reclamation [13], [2], [30], [23], [29], [17], [9], [1]. The main limits of this approach are two fold: the need for information on the applications structure, and the task granularity of the adaptation.

Other quite similar approaches consist in using information of running applications (beyond the graph of tasks) to set the hardware performances according to these. Most of them only consider that hardware is limited to processors and adaptation is limited to frequency scaling, while few consider network, memory and disk as well. The needed information can be given with the application in terms of communication patterns and resource needs [28], [22]. They can also be retrieved for the whole applications or for phases of applications after a pre-characterization process [11], [6], or using a code instrumentation and/or analysis [14], [10], [21], [4]. Application phases can be obtained and used at runtime within a modified code linked to an ad-hoc library [20], [18] or without the need to adapt the application [8], [27], [15]. In these approaches one setting is valid for the duration of one phase.

Classical DVFS (Dynamic Voltage and Frequency Scaling) at kernel level are usually based on CPU utilization. After each time-slot (100ms or less) processor frequency is adapted based on the recent CPU utilization. Frequency is chosen to optimize first performance then power consumption. They are not adapted for HPC workloads as CPU utilization stays most of the time at 100% for such workloads and thus processors are always at maximum frequency. Such approaches encompass classical linux governor (ondemand) or β-adaptation algorithm [15]. In these cases and more generally, sleep states are rarely used as they halt not only the processor but also other processing such as the interactions between the processor and the memory or the network.

A more generic but seldom-used approach is the one of Miser [12] and Green Governor [25]. Authors propose to evaluate if memory or processor is the bottleneck resource at a particular time step (one second or less) and to change the processor frequency accordingly. If the bottleneck is the processor, its frequency is increased, if it is memory, it is decreased. This approach brings energy improvement with limited impact on performance for a large variety of workload without the need for internal information of the application. Our approach is a generalization of these as we propose a framework that consider not only memory but also any resource bottleneck with the same reactivity as the kernel approach. We will illustrate this flexibility by using network resource consumption in the experiment Section. One interesting point is that slack reclamation and bottleneck detection are compatible. Usually the latter can save energy based on a performance reduction budget given by the former.

III. DVFS FOR HPC, TIME- AND ENERGY-WISE

For the evaluation of this research work, applications come from NAS Parallel Benchmark (NPB) [3]. This benchmark provides 7 applications (IS, FT, EP, BT, LU, CG and SP) with workloads being representatives of HPC applications. They exhibit a range of behaviors from an embarrassingly parallel code (EP) to LU decomposition of matrix (LU) with interleaved computations and communications phases. These applications are well studied in the literature and present well known communication and computing patterns.

Benchmark	FT	SP	BT	EP	LU	IS	CG
Time increase (%)	36	69	110	159	96	35	83
Energy increase (%)	-18	2	21	50	16	-19	7

Fig. 1. comparing max frequency (reference, *performance* governor) and min frequency (*powersave* governor)

A. Task-level DVFS

One of the goal of HPC programmers is to use at maximum the raw computing power of the hardware. For instance a classical goal is to overlap communications with computations. If this goal was perfectly achieved, it would mean that reducing raw computing power (by reducing CPU frequency) would increase the makespan (time-to-solution) of HPC applications so much that it would ultimately consume more energy globally.

Energy is computed as the product of makespan and mean power consumed. For instance, if a computer can run at either 1GHz (*powersave* kernel governor) or 2.6GHz (*performance* kernel governor), consuming respectively 150W and 250W, it leads to a computing power ratio of 2.6 and a power consuming ratio of 1.666. With such ratios, if an application is only limited by raw computing power, running at 1GHz would consume nearly 1.56 times more energy than at 2.6GHz.

But real applications are not that simple. To illustrate, Figure 1 shows the behavior of NPB with respect to change in processors' frequency. For instance, for IS benchmark, reducing frequency of all nodes from 2.6GHz to 1GHz leads to 35% increase in execution time only instead of the 160% expected if the limit was only raw computing power. In this scenario, the total energy is reduced by 19%: Indeed, mean power consumption is vastly reduced while time increases only slightly, making the product time-to-solution by mean power more interesting at low frequency than at high frequency for this application.

Those only interested in energy-to-solution metric (energy needed to finish a task) would run IS and FT applications at low frequency, leading to an energy decrease of nearly 20% at the cost of a time penalty of 35%. In HPC environment this time escalation is unacceptable. From these experiments, we understand that reducing frequency has a positive impact on energy-to-solution for IS and FT, while time increases do not correspond to the ratio of frequencies. We conclude that these two applications are not limited only by frequency nor are unrelated to frequency. But since a time increase is observed, raw computing power is, at least partially, a limitation.

B. Phase-level DVFS

Simply choosing for a whole application at which speed to run the processor is inefficient. At a finer grain, an application can be seen in a simplified modeling, in one of the following states:

- Computing
- Communicating (network)
- Disk or memory I/O
- Idle

Benchmark	FT	SP	BT	EP	LU	IS	CG
Time increase (%)	0	-3	-1	1	-2	2	0
Energy increase (%)	0	-3	-1	-1	-2	-1	-1

Fig. 2. Comparing max frequency (reference, *performance* governor) and *ondemand* governor

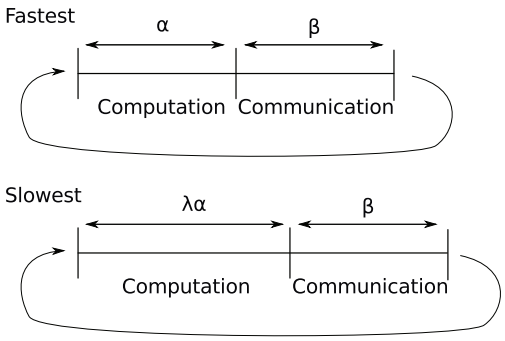

Fig. 3. Classical pattern of HPC application behavior. The length will be $\alpha + \beta$ seconds if using *performance* governor, and $\alpha\lambda + \beta$ seconds if using *powersave*, with $\lambda > 1$. λ is the ratio between maximum and minimum processor frequency.

The Idle case is already covered by classical approaches like the *ondemand* governor. This governor adapts the processor frequency as a function of the load in order to maximize efficiency. Usually HPC applications can not benefit from this governor since these applications' programmers tend to reduce idle times. Figure 2 shows that time and energy are roughly the same for *ondemand* and *performance* on classical HPC benchmarks: Nodes are rarely idle as time-to-solution and energy-to-solution are equals meaning that mean powers are equals. If mean powers are equals, we conclude that most of the time the *ondemand* governor is choosing the maximum frequency.

It is to be noted that most current MPI implementations are actively waiting for packets (polling) and thus applications stay only rarely in Idle state. As explained in the State of the Art, the only relevant case is when a distributed application can be described as a DAG. In this case it is efficient to change processor frequency to the lowest available or to sleep mode when a processor has no tasks. As previously discussed, this article focuses on the case where a task is currently running, in particular on the critical path where performance is of utmost importance.

Disk and memory I/O and network communications are symmetrical: In the following the article will focus on communication without loss of generality. Future work will consist in modeling in the same way other shared buses such as memory or disks.

C. NetSched

We build our approach on the weak hypothesis that during the execution of HPC applications, there is an oscillation between two behaviors, *computing* and *communicating* as shown on Figure 3.

This figure shows successive computing and communication phases cycles. If for one phase the time for computation is α and the time for communication is β while running at full speed, then the same execution at the slowest speed would last $\lambda\alpha$ for the computation part (λ is the ratio between the two speeds) and would not change the communication part. If P_1 is the mean power consumption at full load at full speed, and P_2 is the one at slowest speed, the energies for a cycle are $(\alpha + \beta)P_1$ and $(\lambda\alpha + \beta)P_2$ for respectively fastest and slowest processor speed.

To minimize energy consumption, it is more efficient to stay at maximum speed if

$$(\alpha + \beta)P_1 < (\lambda\alpha + \beta)P_2$$

.

Obtaining α and β at runtime is difficult. We decided to use the application used bandwidth (which is easy to measure) to obtain the ratio α/β, using the relation (where B_m is the maximum bandwidth of the link) :

$$B_w = B_m \frac{\beta}{\alpha + \beta}$$

Merging the two equations, we obtain

$$B_w < \frac{B_m}{\lambda - 1}(\lambda - \frac{P_1}{P_2}) = B_1$$

Hence, when running at full speed, if the current bandwidth is lower than B_1, from the energy consumption point of view it is more interesting to stay at this speed, otherwise it is more interesting to change to a lower speed.

The same reasoning gives B_2 as the threshold over which it is interesting to stay at low speed, and under which it is more interesting to switch to full speed.

$$B_2 = \frac{B_m}{\lambda - 1}(\lambda \frac{P_2}{P_1} - 1)$$

Figure 4 exhibits the incoming byte rate during the execution of each of the 7 applications from the NPB (values shown for one node and one run, horizontally scaled since each application has a different time-to-solution): Applications adapted to low frequency (IS and FT) communicate way more than others. This fits to the presented model: Lowering frequency while communicating decreases the energy.

D. NetSched implementation

In the following we tested our hypothesis that during communications phases frequency can be reduced without impacting computing performances. We designed and implemented *NetSched*, a program that adapts processor frequency during runtime based on monitored network load.

Based on the equations from Section III-C, we consider that if the communication bandwidth is over B_1 (resp. lower B_2) while the processor is at maximum (resp. minimum) speed, it is interesting to reduce (resp. increase) processor speed.

In order to avoid oscillating situations, levels will be respectively 10% over B_1 and 10% below B_2. This creates an hysteresis to improve the stability of the system.

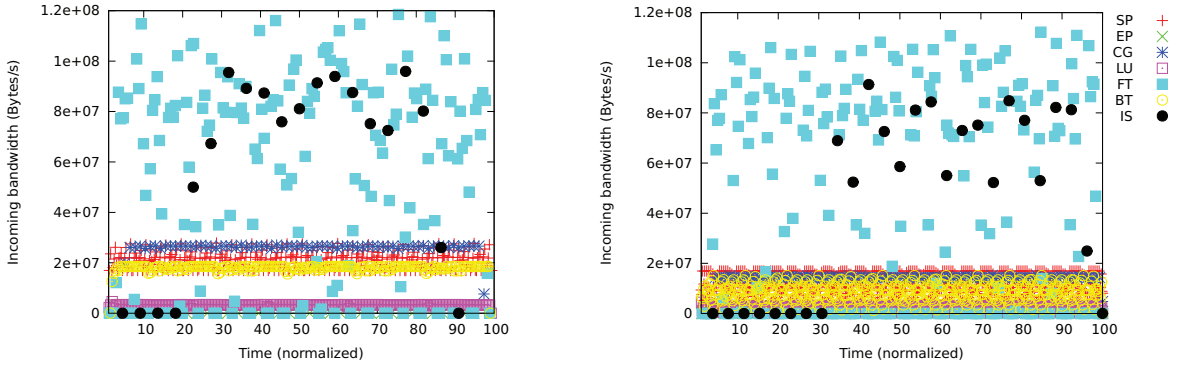

Fig. 4. One example of values of netRECVbyte for each application (left performance, right powersave)

The resulting NetSched algorithm is then (where *IBR* is Incoming Byte Rate):

- Every 10th of second, do:
  - If processor at Slowest frequency and IBR $\leq .9 \times B_1$ $\rightarrow$ Change frequency to Fastest
  - If processor at Fastest frequency and IBR $\geq 1.1 \times B_2$ $\rightarrow$ Change frequency to Slowest

At the moment NetSched is implemented as a Linux daemon using system calls to obtain network bandwidth values as well as changing the frequency. This daemon implementation follows the same philosophy of the Linux kernel governors (such as ondemand). A future step will be the integration of NetSched as a module in the Linux kernel itself.

Compared to the classical state of the art using whole tasks or detecting phases in the application, NetSched behaves like a kernel governor and takes decisions every 100ms.

IV. Experiments

A. Experimental environment

Experiments throughout this article were done on nodes from Toulouse Grid5000 site. Grid5000 [5] is a completely reconfigurable infrastructure where each node can be adjusted to the needs of the experiment.

Each node is connected to a powermeter able to measure power consumption every second. Those monitored nodes are bi Dual-Core AMD Opteron processors (2218) with 8GB of memory and Gigabyte Ethernet card. Processors are DVFS-enabled and can run at frequencies between 1GHz and 2.6GHz.

Nodes run Debian, using 2.6.37 Linux kernel. Since the powermeter sampling frequency is at maximum 1Hz (one sample per second), the power measurement infrastructure for assessing our research work was set up to one second. MPI library is Mpich2. System-level measures are finer grained and by default collected every 100ms.

NPB class C problems are used. NPB classes are related to problem length. Class C problems are solved between half a minute to few minutes on selected nodes. These applications do not use disks during computation, as many HPC applications.

For this batch of experiments, 4 nodes were used, leading to 16 cores experiments. NAS Parallel Benchmark applications are configured to use 16 slots, one per core. Depending on the application, some limits are imposed on the number of slots: Some applications need a power of two, some need a square,... Therefore 16 cores have been used in order to have the same number of slots for the 7 applications.

In this infrastructure, $\lambda = 2.6$ as minimum frequency is 1GHz and maximum frequency is 2.6GHz. $Bm = 1024^3/8$ as the network is Gigabit Ethernet, $P_1 = 280$ and $P_2 = 152$ (in Watts, as measured directly on the nodes).

Using these values, $1.1 \times B_1 \simeq 7.10^7$ and $0.9 \times B_2 \simeq 3.10^7$.

Several policies have been tested on the infrastructure: For all the 7 applications, 4 settings of the governor for the CPU frequency (called DVFS policies in the following) were used, namely performance, powersave, ondemand, and NetSched. For each of these settings, the experiment was repeated 100 times, energy and makespan were monitored, the mean, standard deviation and slope deviation were computed. Therefore, for each application, 400 experiments were used to analyze the behavior of all 4 DVFS policies (including our proposed algorithm NetSched), leading to a total of 2800 experiments. In order to avoid bias the sequence of the 2800 experiments was randomly chosen.

For each block of 100 experiments (meaning for each application and DVFS policy, i.e. 28 blocks), the normalized standard deviation (standard deviation divided by mean value) was computed: 24 values are below 4%, and all 28 are below 7.5%, both for energy and makespan mean values (maximum are for SP and LU applications, shown in Table 5 for the performance policy). Despite the accuracy of the power monitoring infrastructure (sampling rate of our power measures is 1Hz), the standard deviation is still low, especially when considering that the duration of some applications is small in comparison (from 16s to 355s, see Table 5). For the IS application, finishing in 16s, a measure every 1s has an higher impact on accuracy than for SP finishing in 355s.

It can be noted that for some benchmarks like LU, the application consumes less energy with NetSched policy than performance policy and is also faster. Modern processors have

fine-grain control of their thermal budget. When a processor heat exceeds a threshold, its internal frequency is reduced (at hardware level, whatever the operating system governor) in order to let it cool down below a temperature threshold. Using an energy-saving system such as NetSched allows the processor to go faster when needed because it cools down when possible. It results in overall faster runs on several benchmarks while still saving energy (LU, FT, SP) compared to the maximum frequency.

In the same way, we evaluated the normalized slope deviation of the energy and makespan values (computed using least square linear interpolation of values to obtain the slope, values shown are a percentage of the vertical change of this slope for 100 experiments to the mean value). This was done to check if these values were impacted by the sequence of jobs on the hosts (computing machines may heat up and their energy consumption might increase slightly due to this). The average slope of the measured values is almost zero (see Table 5), meaning that the impact of the sequence is negligible. Results are similar for the other DVFS policies than performance.

Performance governor sets the CPU frequency to the maximum for the whole duration of the experiment (2.6GHz on our infrastructure). *Powersave* governor sets the CPU frequency to the minimum for the whole duration of the experiment (1GHz on our infrastructure). Setting the governor to performance or powersave modes has no impact on the operating system since the frequency does not change over time.

The *ondemand* governor increases the CPU frequency to the maximum frequency when the CPU utilization overpass a threshold value. When CPU utilization decreases below this threshold, the governor decreases the frequency step by step: It sets the CPU to run at the next lowest frequency until reaching the lowest possible frequency. The CPU utilization is checked every time-step (100ms on our infrastructure) and the same algorithm is applied to dynamically adjust the CPU frequency to current process load. Like the two previous governors it runs inside the Linux kernel as a module.

NetSched governor has already been described in III-C. At a bird-eye-view it has the same behavior as *ondemand*, adjusting the CPU frequency to the actual behavior of the application. It adapts frequency up to 10 times per second.

The two governors *ondemand* and *NetSched* have an impact on the operating system since a monitoring of the load (for ondemand) or of the IBR (for NetSched) is done and changes in frequency performed. Despite this overhead we will see that results are promising.

B. Experimental results

Figure 6 shows the mean energy consumption on the left, and the makespan on the right. Graphs are normalized so that the reference (100) is set to be the performance policy.

Concerning the energy consumption, we observe that for IS and FT applications, energy consumption is decreased by about 20% with powersave and up to 25% for our NetSched algorithm while ondemand consumes the same energy than performance. For the remaining 5 applications (SP, CG, LU, BT and EP) powersave has higher energy cost (up to 50% increase for EP), while our NetSched has always better energy consumption than performance (except for EP with an increase of less than 1%) and is very close to ondemand (at maximum 3%). We conclude that from an energy point of view, our NetSched algorithm is almost always better than any other policy available. Compared to the ondemand that is also changing the CPU frequency at runtime, our strategy can save up to 25% of energy (see FT application).

Besides energy savings, it is important to check if the duration of the applications are impacted by the policies. For each application, not surprisingly, the powersave policy has a negative impact on the makespan: The frequency is lowered, the CPU takes more time to perform the computation. When the application is mainly composed of raw computation (like the EP application), the time is multiplied by 2.6. Comparing with other policies, we exhibit that our NetSched policy is always better (for 3 applications, and up to 4% for the LU application) or on the same level than the performance and ondemand governor. Only on the IS bench we observe a slight increase in the makespan (less than 5%).

Altogether, we conclude that on the studied applications, our algorithm is beneficial in almost all the cases, with a potential large impact on energy savings (up to 25%), a potential for better makespan on some cases (by a few %) while at worst increasing the energy of less than 1% and the makespan of less than 5%.

Using phase-level DVFS lets NetSched to be in most case more efficient than performance as it reduces processor frequency when processor is the less needed (during communication) while not reducing it when it would impact makespan. Makespan is then globally stable but mean power consumption is reduced, so energy-to-solution is reduced.

Compared to powersave, NetSched has smaller time-to-solution even if mean power is higher: While doing only communications their mean power consumptions are the same, but NetSched benefits from the fact that doing raw computations at maximum frequency is more energy-efficient than at minimum frequency.

Finally the comparison with ondemand has large similarity with performance as ondemand scarcely manages to reduce frequency for the applications at hand, as explained earlier.

C. Validation

Previous experiments show that for efficient HPC code (i.e. without idle time) using network bandwidth for adjusting the CPU frequency leads to improvements. In a more generic case, using at the same time *load* (ondemand-like), network (NetSched-like), and IO information would certainly lead to an energy efficiency improvement greater than the one obtained solely with NetSched. Currently most DVFS techniques take only load into account. Figure 7 shows energy consumption of NPB using performance as a reference. Without entering in the details of all governors tested, we notice that all of them (other than performance and powersave) are using only the load of the system to manage dynamically the CPU frequency. Most of

Application	FT	SP	BT	EP	LU	IS	CG
Makespan (seconds)	167	355	210	33	239	16	71
Normalized Standard deviation (makespan)	1.89	6.27	3.06	1.37	7.82	4.19	2.21
Normalized Slope deviation (makespan)	0.44	1.16	-0.24	-0.09	1.96	-2.91	0.41
Energy (Kilo Joules)	170	366	222	33	249	17	75
Normalized Standard deviation (energy)	1.93	6	2.81	1.63	7.33	3.34	1.98
Normalized Slope deviation (energy)	0.91	0.97	-0.15	0.6	2.29	-1.54	0.09

Fig. 5. Mean energy consumption, makespan, standard deviation, slope deviation for performance policy.

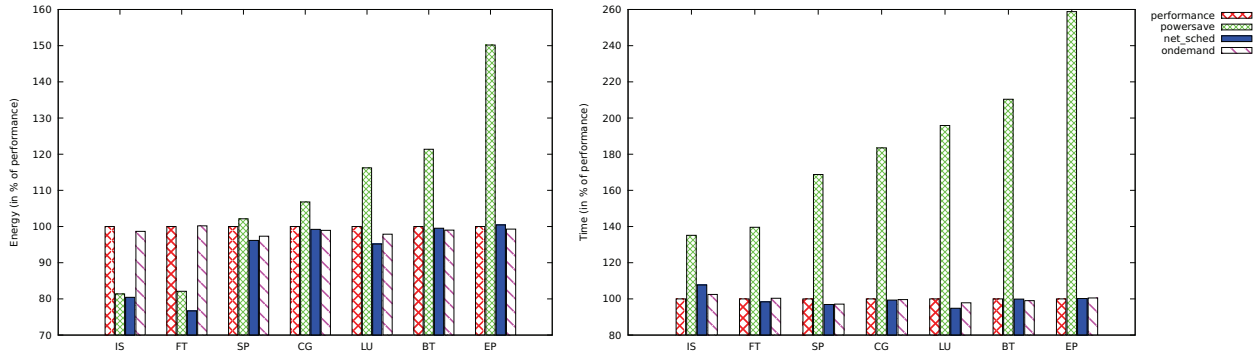

Fig. 6. Mean energy consumption (left) and makespan (right) of DVFS policies. Please note that y-axes do not start at 0.

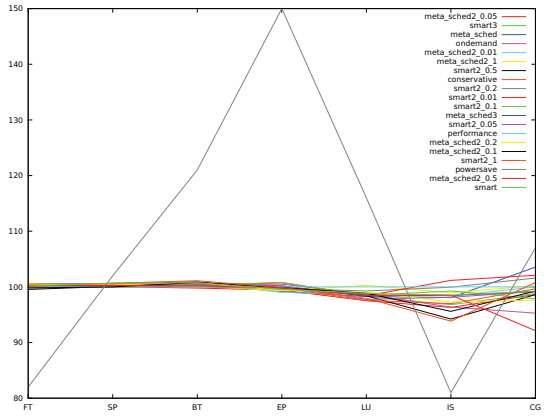

Fig. 7. Energy consumption (energy-to-task) for several schedulers that base their decision only on load. Reference is performance. Classical linux performance and conservative governors are present.

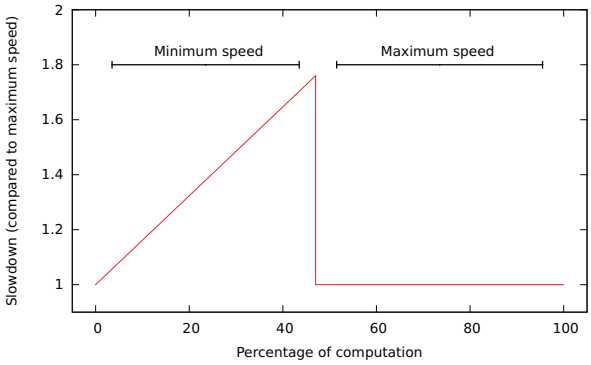

Fig. 8. Theoretical slowdown of applications as a function of computation percentage using NetSched. Worst case is reached at about 45% of computation (using values of Section IV-A).

them behave as performance as they do not detect any idle time (ondemand and conservative are present). Next HPC governors will need an holistic view in order to achieve maximal gains. Current implementation of NetSched uses only two processor frequencies, but the model can be applied to any number of frequency levels using the same methodology.

One particularly interesting point is that using the model designed in Section III-C, the reduction in energy consumption should be correlated with an increase in execution time as shown on Figure 8. In real cases, Figure 6 shows that most of the time this increase is absent but for one case. The case of FT is particularly interesting as it shows the best improvement in energy consumption without performance penalty. This property is linked to the communication pattern. A significant slowdown would appear only if the application behavior is a mix of communication and computation close to the point NetSched changes the frequency from minimum to maximum (at about 45% using our environment, seen on Figure 8). This limit is actually when $(\alpha + \beta)P_1 = (\lambda\alpha + \beta)P_2$. In this case the slowdown would be P_1/P_2 (i.e. 1.8 in our experimental environment, 1.6 taking into account the hysteresis). As seen

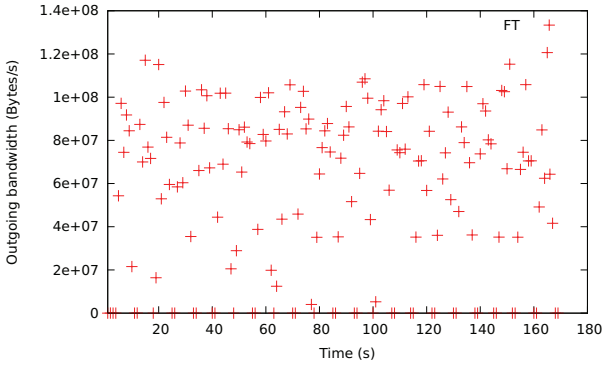

Fig. 9. Communication pattern during a run of FT using *performance* governor

in Figure 9, FT does not follow any pattern and is rarely in this region, hence not suffering much from this slowdown.

Furthermore it must be noted that these results were obtained using 16 slots. A larger scale would increase the ratio communication over computation. In this case NetSched would exhibit even more savings, following this ratio.

V. Impact on HPC centers

System administrators of HPC centers are confronted with a dilemma: On the one hand their users require more and more computing power (and often obtain budgets for the hardware) and are not ready to trade any performance loss for energy concerns; On the other hand they have to handle two problems: (1) their operational cost is limited by external factors (like financial stress) where the operational cost includes costs for the electricity and costs for the staff. (2) their energy provider is capping their potential instantaneous power to the physical limits of the power plants or the electrical wires bringing the energy to the data center.

In this perspective, our proposal has several merits:

- Reducing the energy consumption of HPC applications leads to lower operational energy costs. Given a budget limit, this means that more computations can be done for the same price, especially when the machines are managed on a pay-per-use perspective.
- It allows to make more computations with the same physical limit on maximum power usage (power capping), this gain will depend on the particular application.
- The operating system is adjusting automatically to the actual applications on the machines, without the loss of performances from a user point of view, and without any additional staff commitment.
- It does not involve any change in the HPC application.
- It is fully complementary with other approaches at the hardware level (taking more energy efficient hardware and infrastructure) and at the middleware level: Indeed, acting at the machines operating system level allows our approach to be transparent from its environment. Also, every watt saved at the lower level of an infrastructure has an indirect impact on the other levels as well (including the cooling infrastructure).

The goal of exascale computing for a 20 MW power budget is achievable only when all the leverages are used and combined together. This improvement is not sufficient, but is one of the pieces needed to achieve it. Many HPC centers are not prepared for higher power demands (on the energy provider point of view) and reducing every watt is a necessary step to be able to handle more and more complex applications and computing power. We do believe that our strategy is a stepping stone towards this goal.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see https://www.grid5000.fr)

VI. Conclusion and perspectives

The main contributions of this article are two-fold. First it shows that having an holistic view of the system instead of relying only on load leads to improve energy efficiency without sacrificing performances for HPC applications. Second it provides the model and algorithm of a decision process reducing energy consumption based on network utilization. Experiments on NAS Parallel Benchmarks show that large energy savings can be achieved (up to 25%) without impacting the performances of the applications in most of the cases (even increasing them on 3 cases, up to 4%) with only one case where the performance is decreased (less than 5%).

The proposed method is complementary and compatible with other methods proposed for HPC applications based on global applications characteristics (using expected slacks in DAG, synchronizing different tasks,...) since it runs at the lower level of the operating system.

The validation of our approach exhibited in this article must be confronted with several other benchmarks and real applications, in order to pass from an application to a behavioral perspective. The NetSched algorithm is very simple at the moment while achieving already good results. We want to check several directions to extend its scope and to limit the rare situations when slight increases in energy or time are witnessed. Among those, we will first merge it with the ondemand governor in order to be able to manage idle time and network usage. We will also increase the number of possible frequencies, from the two currently used in our proof of concept to an arbitrary number. The resulting governor will be able to manage larger use-cases, not only limited to well-written HPC applications. The goal of future experiments is to run large scale applications. Then, other components will be included, first one being an IO module and a memory module following the same idea than the network one.

The NetSched algorithm is currently implemented in user-space and will benefit from being in kernel-space. It would reduce its footprint by reducing the number of system calls.

Also, constants B_1 and B_2 are easy to compute but work is still to be done to evaluate them automatically or to adjust them on the fly.

Finally merging NetSched with a slack reclamation algorithm is the next step. Depending on the constants B_1 and B_2 it is possible to model the slowdown and thus to optimize even more energy consumption outside of the critical path.

An interesting perspective will also be to experiment on heterogeneous architecture such as big.LITTLE in order to generalize our concept of running platform. It will done by considering a parallel between different architectures and different frequencies in our current model.

REFERENCES

[1] Pedro Alonso, Manuel F. Dolz, Francisco Igual, Rafael Mayo, and Enrique S. Quintana-Ortí. Dvfs-control techniques for dense linear algebra operations on multi-core processors. In *International Conference on Energy-Aware High Performance Computing*, September 2011.

[2] Guillaume Aupy, Anne Benoit, Fanny Dufossé, and Yves Robert. Reclaiming the energy of a schedule: models and algorithms. *Concurrency and Computation: Practice and Experience*, 25(11):1505–1523, 2013.

[3] David Bailey, Tim Harris, William Saphir, Rob van der Wijngaart, Alex Woo, and Maurice Yarrow. The nas parallel benchmarks 2.0. Technical report, NAS Technical Report NAS-95-020, NASA Ames Research Center, Moffett Field, CA, 1995.

[4] Denis Barthou, Andres Charif Rubial, William Jalby, Souad Koliai, and Cedric Valensi. Performance tuning of x86 openmp codes with maqao. In *Parallel Tools Workshop*, pages 95—113, Dresden, Germany, September 2009. Springer-Verlag.

[5] Franck Cappello, Eddy Caron, Michel J. Daydé, Frédéric Desprez, Yvon Jégou, Pascale Vicat-Blanc Primet, Emmanuel Jeannot, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Benjamin Quétier, and Olivier Richard. Grid'5000: a large scale and highly reconfigurable grid experimental testbed. In *GRID*, pages 99–106, http://www.grid5000.fr/, 2005. IEEE.

[6] Franck Cappello, Amina Guermouche, and Marc Snir. On communication determinism in parallel hpc applications. In *Computer Communications and Networks (ICCCN), 2010 Proceedings of 19th International Conference on*, pages 1–8, August 2010.

[7] Davide Careglio, Georges Da Costa, Ronen Kat, Avi Mendelson, Jean-Marc Pierson, and Yannakis Sazeides. Hardware leverages for energy reduction in large scale distributed systems. Technical report, IRIT, 2010.

[8] Georges Da Costa and Jean-Marc Pierson. Characterizing applications from power consumption: A case study for HPC benchmarks. In *International Conference on ICT as Key Technology for the Fight against Global Warming (ICT-GLOW), Toulouse, 29/08/2011-02/09/2011*. Springer, 2011.

[9] Maja Etinski, Julita Corbalan, Jesus Labarta, Mateo Valero, and Alex Veidenbaum. Power-aware load balancing of large scale mpi applications. In *IPDPS '09: Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, 2009. IEEE Computer Society.

[10] Vincent W. Freeh and David K. Lowenthal. Using multiple energy gears in mpi programs on a power-scalable cluster. In *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, PPoPP '05, pages 164–731, New York, NY, USA, 2005. ACM.

[11] V.W. Freeh, Feng Pan, N. Kappiah, D.K. Lowenthal, and R. Springer. Exploring the energy-time tradeoff in mpi programs on a power-scalable cluster. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, page 4a, april 2005.

[12] Rong Ge, Xizhou Feng, Wu-chun Feng, and Kirk W Cameron. Cpu miser: A performance-directed, run-time system for power-aware clusters. In *Parallel Processing, 2007. ICPP 2007. International Conference on*, pages 18–18. IEEE, 2007.

[13] Tom Guérout, Thierry Monteil, Georges Da Costa, Rodrigo Neves Calheiros, Rajkumar Buyya, and Mihai Alexandru. Energy-aware simulation with DVFS. *Simulation Modelling Practice and Theory*, 39(1):76–91, Dec. 2013.

[14] Yoshihiko Hotta, Mitsuhisa Sato, Hideaki Kimura, Satoshi Matsuoka, Taisuke Boku, and Daisuke Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *IPDPS*. IEEE, 2006.

[15] Chung-hsing Hsu and Wu-chun Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, SC '05, pages 1–, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Wei Huang, Jiuxing Liu, Bulent Abali, and Dhabaleswar K. Panda. A case for high performance computing with virtual machines. In *Proc. of the 20th Annual Intl. Conf. on Supercomputing*, pages 125–134, 2006.

[17] N. Kappiah, V.W. Freeh, and D.K. Lowenthal. Just in time dynamic voltage scaling: Exploiting inter-node slack to save energy in mpi programs. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, page 33, nov. 2005.

[18] Dong Li, B.R. de Supinski, M. Schulz, K. Cameron, and D.S. Nikolopoulos. Hybrid mpi/openmp power-aware computing. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1 –12, april 2010.

[19] Yongpeng Liu and Hong Zhu. A survey of the research on power management techniques for high-performance systems. *Softw. Pract. Exper.*, 40:943–964, October 2010.

[20] T. Minartz, T. Ludwig, M. Knobloch, and B. Mohr. Managing hardware power saving modes for high performance computing. In *Green Computing Conference and Workshops (IGCC), 2011 International*, pages 1 –8, july 2011.

[21] Thomas Panas, Dan Quinlan, and Richard Vuduc. Tool support for inspecting the code quality of hpc applications. In *Proceedings of the 29th International Conference on Software Engineering Workshops*, pages 182–, Washington, DC, USA, 2007. IEEE Computer Society.

[22] Jean-Marc Pierson and Henri Casanova. On the Utility of DVFS for Power-Aware Job Placement in Clusters. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par, Bordeaux, 31/08/2011-02/09/2011*, number 6852 in LNCS, pages 255–266, http://www.springerlink.com/, septembre 2011. Springer-Verlag.

[23] Nikzad Babaii Rizvandi, Javid Taheri, and Albert Y. Zomaya. Some observations on optimal frequency selection in dvfs-based energy consumption minimization. *J. Parallel Distrib. Comput.*, 71(8):1154–1164, August 2011.

[24] Barry Rountree, David K. Lownenthal, Bronis R. de Supinski, Martin Schulz, Vincent W. Freeh, and Tyler Bletsch. Adagio: Making dvs practical for complex hpc applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 460–469, New York, NY, USA, 2009. ACM.

[25] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive dvfs. In *Proceedings of the 2011 International Green Computing Conference and Workshops*, IGCC '11, pages 1–8, Washington, DC, USA, 2011. IEEE Computer Society.

[26] Anand Tikotekar, Geoffroy Vallée, Thomas Naughton, Hong Ong, Christian Engelmann, and Stephen Scott. An analysis of hpc benchmarks in virtual machine environments. In Eduardo Cesar, Michael Alexander, Achim Streit, Jesper Trff, Christophe Crin, Andreas Knpfer, Dieter Kranzlmller, and Shantenu Jha, editors, *Euro-Par 2008 Workshops - Parallel Processing*, volume 5415 of *Lecture Notes in Computer Science*, pages 63–71. Springer Berlin / Heidelberg, 2009.

[27] Ghislain Landry Tsafack Chetsa, Laurent Lefvre, Jean-Marc Pierson, Patricia Stolf, and Georges Da Costa. Harnessing Performance Counters for Predicting and Improving Energy Performance of HPC Systems. *Future Generation Computer Systems*, 10.1016/j.future.2013.07.010:1–18, aot 2013.

[28] Akshat Verma, Puneet Ahuja, and Anindya Neogi. Power-aware dynamic placement of hpc applications. In Pin Zhou, editor, *ICS*, pages 175–184. ACM, 2008.

[29] Lizhe Wang, Gregor von Laszewski, Jay Dayal, and Fugang Wang. Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 368–377, Washington, DC, USA, 2010. IEEE Computer Society.

[30] Weixun Wang, Sanjay Ranka, and Prabhat Mishra. Energy-aware dynamic slack allocation for real-time multitasking systems. *Sustainable Computing: Informatics and Systems*, 2(3):128–137, 2012.