Simulation Modelling Practice and Theory 39 (2013) 76–91

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

Energy-aware simulation with DVFS

Tom Guérout [a,b,c,*], Thierry Monteil [a,c], Georges Da Costa [b], Rodrigo Neves Calheiros [a], Rajkumar Buyya [e], Mihai Alexandru [a,d]

[a] CNRS, LAAS, 7 Avenue du Colonel Roche, F-31400 Toulouse, France
[b] IRIT/Toulouse University, 118 Route de Narbonne, F-31062 Toulouse Cedex 9, France
[c] Univ de Toulouse, INSA, LAAS, F-31400 Toulouse, France
[d] Univ de Toulouse, INP, LAAS, F-31400 Toulouse, France
[e] Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 30 November 2012
Received in revised form 29 April 2013
Accepted 30 April 2013
Available online 4 June 2013

Keywords:
Grid computing
Cloud computing
Simulation
Energy-efficiency
DVFS

ABSTRACT

In recent years, research has been conducted in the area of large systems models, especially distributed systems, to analyze and understand their behavior. Simulators are now commonly used in this area and are becoming more complex. Most of them provide frameworks for simulating application scheduling in various Grid infrastructures, others are specifically developed for modeling networks, but only a few of them simulate energy-efficient algorithms. This article describes which tools need to be implemented in a simulator in order to support energy-aware experimentation. The emphasis is on DVFS simulation, from its implementation in the simulator CloudSim to the whole methodology adopted to validate its functioning. In addition, a scientific application is used as a use case in both experiments and simulations, where the close relationship between DVFS efficiency and hardware architecture is highlighted. A second use case using Cloud applications represented by DAGs, which is also a new functionality of CloudSim, demonstrates that the DVFS efficiency also depends on the intrinsic middleware behavior.

1. Introduction

Simulators are now commonly used to study the energy consumption of data centers [1,2]. Indeed, the increasing use of data centers and the high amount of energy demanded by them, make the analysis of their energy consumption extremely important. Several metrics to evaluate their efficiency are widely used (PUE, ERE) [3] and research is ongoing to find new ways to reduce energy consumption and improve energy efficiency metrics. This leads to improvement and/or creation of new algorithms that require extensive testing phases to validate their effectiveness.

Each phase of development and testing of these algorithms are very long due to the number of attempts required to check and improve their performance. Real platforms can be used on these validation phases, but it involves a significant preparation time, and physical measurements are not always possible or easy to acquire and reproduce, depending on the available equipment. Hence, simulators are commonly used in this area.

The article focuses on the comparison between energy-aware simulations and real experiments, especially using DVFS (Dynamic Voltage and Frequency Scaling). This comparison handles the problem of how to choose a simulator that offers

* Corresponding author at: CNRS, LAAS, 7 Avenue du Colonel Roche, F-31400 Toulouse, France. Tel.: +33 561336924.
E-mail addresses: tguerout@laas.fr (T. Guérout), monteil@laas.fr (T. Monteil), dacosta@irit.fr (G. Da Costa), rnc@unimelb.edu.au (R. Neves Calheiros), raj@csse.unimelb.edu.au (R. Buyya), malexand@laas.fr (M. Alexandru).

1569-190X/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2013.04.007

some energy-aware modeling tools, how to improve it, how to validate its new functionalities based on energy consumption and finally how to use it to conduct accurate simulations.

The use of DVFS is common in HPC and Grid systems, but not commonly developed in simulators. For this reason, this article initially presents a DVFS integration and validation phase. This phase demands many detailed steps, from the power model study to how to define an application scenario to use in real experiments and simulations in order to achieve precise validation.

This article is organized as follows. Sections 2 and 3 present a state of the art on energy models, energy-aware tools, simulators, DAG (Directed Acyclic Graph) and DVFS. Sections 4.1 and 4.2 describe implementation of DVFS and DAG that have been added to CloudSim. Section 5 presents the validation phase of the DVFS implementation in the simulator. In Section 6, two use cases are detailed and a comparison between simulations and real experiments is established. The first one uses a parallel Grid application described in Section 6.1 and the second one presents Cloud simulation using both DVFS and DAG in Section 6.2. Finally, Section 7 concludes the article and presents ideas for future work.

2. State of the art

This section presents state of the art concepts and tools that are relevant for energy-aware studies. In simulation, one of the main challenges is how to develop a power model that fits as best as possible the behavior of a real host. In the domain of real life tools for energy efficiency, virtualized environments provide several techniques for reducing energy consumption, through virtual machines and host management. These solutions are listed in this section and used on both simulations and experiments. Moreover, a list of widely used energy-aware simulators actually available is presented. The last point is about DAGs which, are commonly used to represent relevant applications in the domain of scientific computing.

2.1. Power models

Rivoire et al. [4] established a comparison list between many different energy models. Two categories are defined: comprehensive models and high-level models called *black-box models*. Comprehensive CPU models were proposed by Joseph and Martonosi [5] and by Isci and Martonosi [6]. These models achieve high accuracy in terms of CPU power consumption and rely on specific details of the CPU micro-architecture. High-level models are more general, and disregard specific low-level details of the operation and characteristics of the modeled infrastructure. Nevertheless, they increase portability of the model and simulation speed. These simple models sacrifice the accuracy of computation, but do not require detailed knowledge of the architecture of processors used.

In this article, the high-level approach is applied. The linear power model used at fixed frequency is:

$$P_{TOT} = (1 - \alpha)P_{CPUIdle} + \alpha P_{CPUFull}$$

where α is the CPU load and $P_{CPUIdle}$ and $P_{CPUFull}$ are the power consumed by the CPU at 0% and 100% of utilization, respectively. Alternatively, more accurate models could be used as in the work by Da Costa et al. [7].

2.2. Energy-aware tools

Energy-aware tools are solutions that can be used at different levels, allowing the minimization of the power consumption of each host in a data center. Some solutions act on hosts or on the network. In this article, only the host level is addressed and this section presents three common energy-aware tools that require that all used hosts utilize system virtualization technology.

The first solution, called ON/OFF method, turns off hosts underutilized (compared to a threshold) and switches them on again if necessary. All processes running on an underutilized host are moved to other hosts, and then the underutilized host is switched off. Conversely, when all the hosts are over-utilized and the demand is high, one or more hosts are switched on again. This process has only one goal, which is to reduce the number of switched-on hosts on a given time.

As described above, it is sometimes necessary to use migration mechanisms [8] to move processes between hosts. Migration allows the transfer of a running virtual machine (and its entire environment) from one host to another. These transfers are not free in terms of energy consumption, because every change requires some time where data is transferred between source and destination. It is also important to take into account the total cost of such action. This technique allows the release of some hosts and their deactivation, at the same time that the remaining running hosts can be used at their maximum potential (consolidation).

Finally, DVFS (Dynamic Voltage and Frequency Scaling) [9] can dynamically change the voltage and frequency of a CPU of a host according to its load. In the Linux kernel, DVFS can be activated in five different modes: *Performance*, *PowerSave*, *UserSpace*, *Conservative*, and *OnDemand*. Each mode has a *governor* to decide whether the frequency must be changed (increased or decreased) or not. Three of these five modes use a fixed frequency: *Performance* uses the CPU at its highest frequency, *PowerSave* uses the lowest frequency and the *UserSpace* mode allows the user to choose one of the available frequencies for the CPU. The two last modes, *Conservative* and *OnDemand*, have a dynamic behavior. It means that the CPU frequency can vary over time regarding the CPU demand. The *governors* of these two modes work with thresholds (one or two) and periodically

check whether the CPU load is under (or over) these thresholds before making the decision to change the CPU frequency. The *Conservative governor* works with an *up_threshold* and a *down_threshold*. When the CPU load is above the *up_threshold* the frequency is increased, and when the CPU load is below the *down_threshold* the CPU frequency is decreased. This mode is progressive, and each CPU frequency change is performed at one step step using all available frequencies. The *OnDemand* mode uses only one threshold and favors system performance by directly setting the fastest frequency when the CPU load exceeds the threshold. A decreasing CPU frequency, performed at steps as in the *Conservative* mode, is performed if the CPU load stays below the threshold for a predefined amount of time.

A lower frequency, which implies a weaker voltage, reduces CPU power consumption but also slows down the CPU computation capacity. Regarding the time spent with I/O operations, the efficiency of DVFS depends on the system architecture. In some cases, it also slows down the I/O time when DVFS affects the frequency of the FSB (Front Side Bus). If a frequency change only affects the multiplier between the FSB and the CPU frequency, I/O performance is not affected.

2.3. Simulators

SIMGRID [10] is a simulator that provides basic functionality for simulation of distributed applications in heterogeneous distributed environments. Its use is well suited for evaluation of heuristics, prototyping, development, and improvement of grids applications.

GroudSim [11] offers event-simulation tools for grid computing and cloud computing. The main areas of simulation are file transfers, the cost of resources and the calculation of operating costs. It can be used with a package containing information about the probability of hardware failure. In this case, the event module can detect errors occurring on the host or on the network and initiate a process of reconfiguration of the concerned entities.

GSSIM [12,13] is a simulator that allows the study of scheduling policies with a flexible description of the architecture and interactions between modules. Standard formats are accepted for description of workflows, such as *Standard Workload Format* (SWF) and *Grid Workload Format* (GWF). An XML file can be defined to include more detailed information (e.g., time constraints). GSSIM allows virtualization and also integrates energy models and an accurate representation of the resource usage.

GreenCloud [14] is an extension of the network simulator ns2 [15]. It provides a detailed modeling and analysis of the energy consumed by the elements of the network servers, routers and links between them. In addition, it analyzes the load distribution through the network, as well as communications with high accuracy (TCP packet level). In terms of energy, GreenCloud defines three types of energy: calculation (CPU), communications, and physical computing center (cooling system), and includes two methods of energy reduction: DVS (Dynamic Voltage Scaling) to decrease the voltage of switches and DNS (Dynamic Network Shutdown) that allows to shut down switches when it is possible.

CloudSim [16] is a toolkit for modeling and simulation of Infrastructure as a Service (IaaS) cloud computing environments. It allows users to define the characteristics of data centers, including number and characteristics of hosts, available storage, network topology, and patterns of data centers' usage. It allows the development of policies for placement of virtual machines on hosts, allocation of cores to virtual machines, and sharing of processing times among applications running in a single virtual machine. Energy modeling in CloudSim allows termination of hosts for energy saving, virtual machine migration, and integration of energy models. The application layer is managed by *brokers*, which represent users of the cloud infrastructure, that request creation of virtual machines in the data center. A *broker* can simultaneously own one or more virtual machines, which execute application tasks. Virtual machines are kept operating while there are tasks to be executed, and they are explicitly deallocated by the *broker* when all the tasks finish. Capacities of both virtual machines and hosts' CPUs are defined in MIPS (Million Instructions Per Second). Tasks, called *cloudlets*, are assigned to virtual machines and defined as the number of instructions required to their completion.

Other commercial energy-aware simulators can be found in [1,2].

None of the available open source simulators contains a model for DVFS. Therefore, a model for this technique has been developed and incorporated in the CloudSim simulator, because this simulator contains abstractions for representing distributed Cloud infrastructures and power consumption.

2.4. Directed acyclic graph

Direct Acyclic Graph (DAG) or workflow applications are a popular model for describing scientific distributed applications. Such applications are composed of a number of tasks (which are represented as nodes in the graph) that have data dependency relationship between them (modeled as the directed vertices from the *parent* task to the *child* task). Execution of a *child* task cannot start until all its *parents* completed their execution and generated their output. Indeed, the outputs are required by the *child* task so as to use them as inputs for its execution.

Several algorithms were developed for execution of DAGs in clusters [17,18], grids [19,20], and Clouds [21,22]. However, these existing algorithms typically focus in reducing execution times (in clouds and grids) or the execution time and/or execution cost (in the case of Clouds).

Recently, works were developed towards energy-efficient execution of workflows in clusters via DVFS [23,24]. However, more research is necessary in this area in order to evaluate the current approaches against well known and widely used workflow applications [25]. Therefore, the simulation model introduced in this article can be a valuable tool for developers

of such algorithms, as it allows them to obtain quicker turnaround time during development of new algorithms, as it is demonstrated in a case study in Section 6.2.

3. DVFS in the Linux kernel

DVFS is enabled in the Linux kernel since 2001. It can be used by installing the *cpufrequtils* package and its documentation gives detailed informations about all modes behavior. It is located in the kernel documentation (*linux-x.y.z/Documentation/cpu-freq*) and all DVFS sources code are also available in this folder: *linux-x.y.z/drivers/cpufreq*.

This DVFS module allows the system to switch between the five different modes as per the user's convenience, by selecting their *governor*. Once the DVFS package is installed and configured, information about the system and the *governors* can be obtained:

- CPU(s) affected by the DVFS;
- Available *governors* (one for each mode);
- Available frequencies of the host. These frequencies are directly defined by the type of host and CPU;
- Maximum and Minimum available frequencies are also stored in separated files. In some cases it is useful to set them with specific values;
- Transition latency: meaning the minimum interval time the system check the CPU load;
- The current *governor* in use.

Each DVFS mode can be configured in different ways by setting specific parameters values.

- Sampling rate: defines the minimum time interval between two frequency changes;
- Thresholds: The *OnDemand* and *Conservative* modes compare the current CPU load with predefined thresholds. By setting custom values, the user can adapt their behavior.

The following information about statistics usage of each mode can also be obtained:

- The time spent on each frequency;
- The total frequency transitions;
- A detailed table of frequency changes, with the number of transitions between one frequency to the others.

4. New functionalities in CloudSim

In this section, the features developed and incorporated to the CloudSim simulator in order to support modeling and simulation of DVFS and workflow applications are described.

4.1. DVFS in CloudSim

The core features for DVFS simulation were added to a new package, called DVFS. In this package (Fig. 1), *governors* of the five modes of DVFS, as they are present in the Linux kernel and described in Sections 2.2 and 3, were implemented. Their role is to determine if the frequency of the CPU must be changed and their decision is directly related to their intrinsic decision rule. In the simulator, a frequency change directly impacts the capability of CPUs, measured in MIPS.

The use of DVFS directly affects the performance of the CPU capacity (and hosts), which are subject to regular changes during simulations. This also involves changing the way the simulator handles the placement and management of virtual machines. For example, if the system decides to reduce the frequency, the sum of capacities of all virtual machines (VMs) may temporarily exceed the maximum capacity of the host. In this case, the size of each VM must temporarily be adjusted downward in proportion to the capacity of the new host. The same situation occurs when one or more new virtual machines have to be added to a host already in use. If the sum of capacities of the virtual machines already running in the host plus the capacities of new virtual machines created exceeds the capacity of the host, the size of all virtual machines has to be decreased before adding the new virtual machines to the host.

Also, when part of the capacity of a host is released, capacities of virtual machines still active may increase. This event occurs when a virtual machine has finished its execution or when the CPU frequency is increased. In this case, the capacity of all the remaining virtual machines are scaled up regarding the free capacity of the host, while taking care to do not exceed its maximum capacity.

The inclusion of DVFS in CloudSim required changes in the management of events performed by the simulation core. In fact, the *PowerSave* mode induces a delay in the execution of tasks as the CPU operates at its lower frequency. Under these conditions, the evolution of sequential events should be directly linked with the end of each event. The static description of events (Fig. 2a) is no longer valid because it does not take into account the execution delay (illustrated in Fig. 2b). Thus, a new option has been added to enable creation of events at the end of the execution of a *broker*, which means that all *cloudlets*

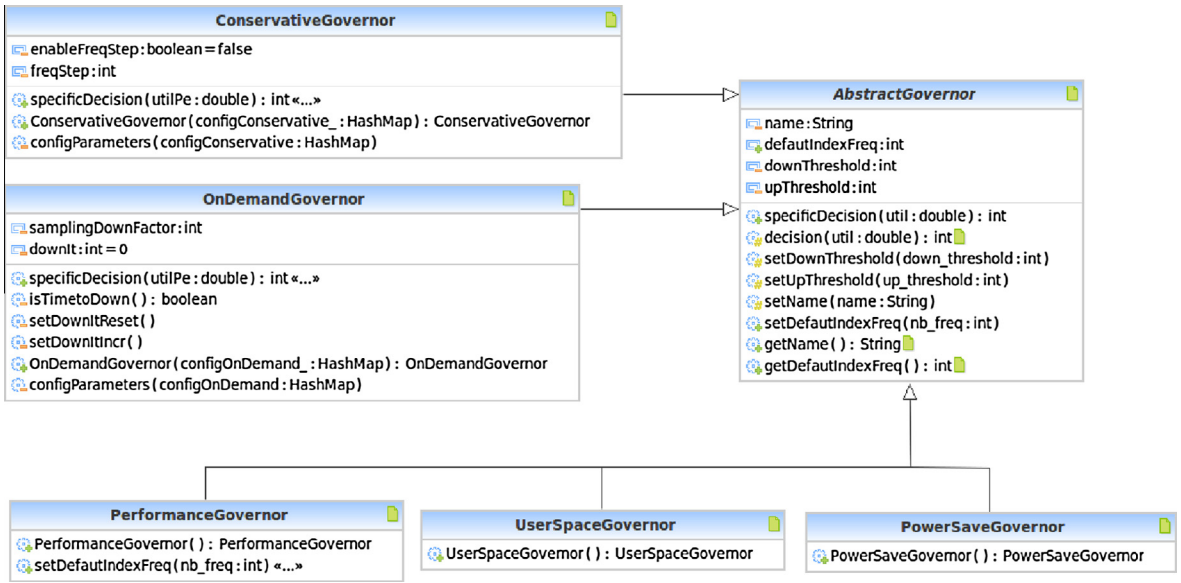
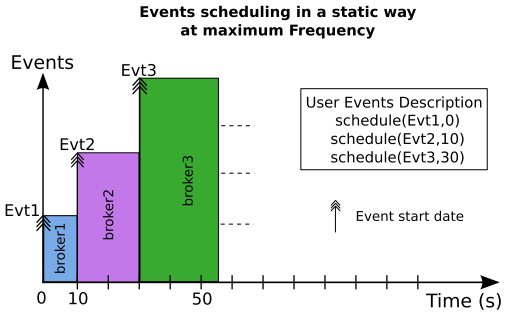
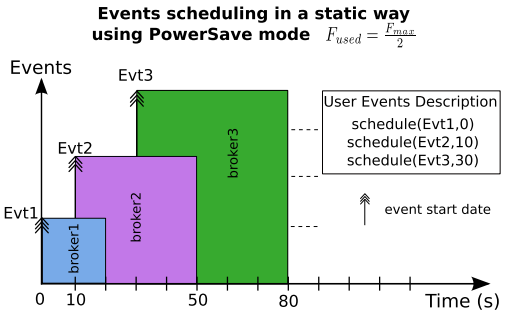


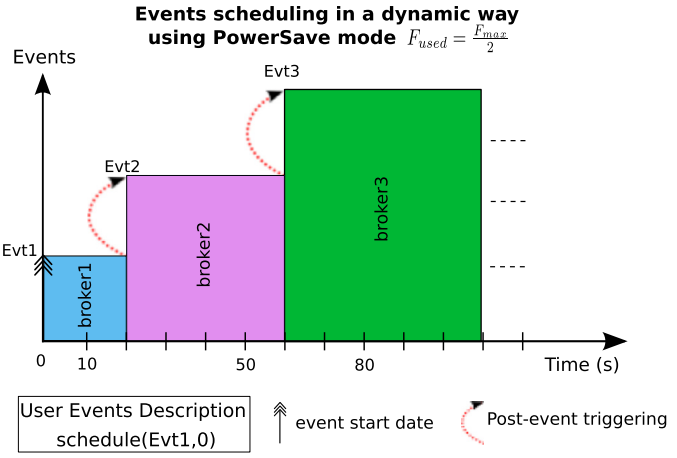
Fig. 1. UML diagram of DVFS package in CloudSim.



(a) Normal static events functioning



(b) Static events using *PowerSave* mode



(c) Dynamic triggering of events using *PowerSave*

Fig. 2. (a) Shows an example of a base CloudSim event scheduling with a static description of events start dates at maximum frequency (no execution delay in this case). (b) Shows an example of a base CloudSim event scheduling with a static description of events start dates, executed using *PowerSave* mode ($F_{used} = \frac{F_{max}}{2}$), which implies delay in *brokers* execution. In this case it is easy to see that *brokers* execution sequence is not correct because of static event description does not take into account generated delays (leading to a superposition of *brokers* execution). (c) Shows an example of dynamic trigger of events. This new option added in CloudSim allows to use low frequencies that incur delays during *cloudlet* execution without disturbing the results of the simulation. The first event of the simulation is defined at $t = 0$ and then *brokers* defined with a *post-event* continue the events sequence.

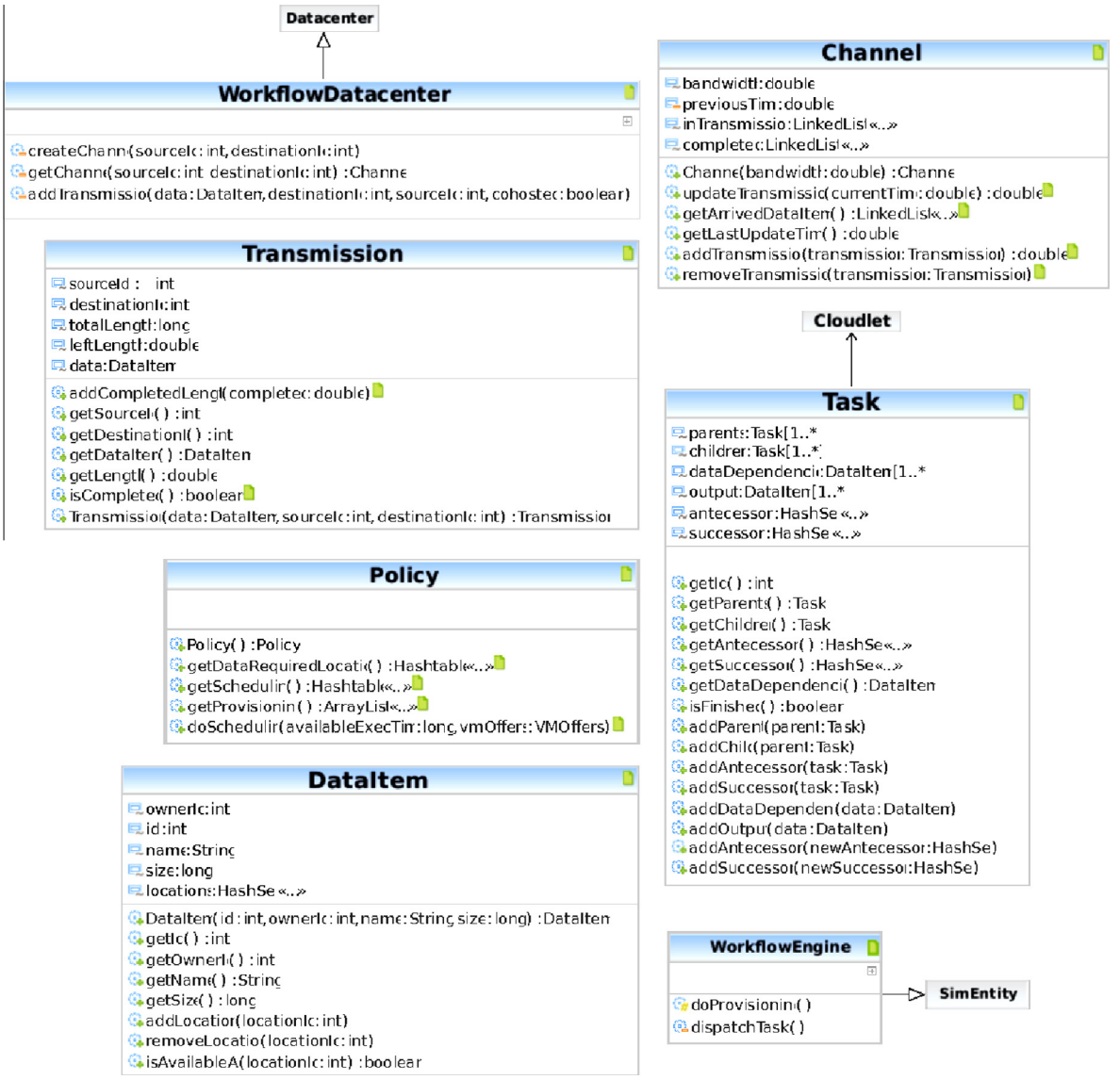


Fig. 3. Class diagram of support for modeling and simulation of power-aware DAGs in CloudSim.

contained in the *broker* have finished their execution before the start of the next *broker*. When a *broker* is instantiated, it can contain the specification of a *post-event* that triggers a new event which starts the execution of a new sequence of *cloudlets*.

The last change in the simulator is the energy model. Indeed, the energy model used depends on the host to be modeled in the simulation. Model definition used in this experiment is described in Section 5.2.

4.2. DAGs in CloudSim

Model and simulation of execution of power-aware DAGs in CloudSim was enabled with a series of extensions to the basic CloudSim objects in order to make them aware of the dependencies between tasks, as well as to account for data transfer between virtual machines. This implementation is independent from, and was developed concurrently with, the implementation by Chen and Deelman [26]. This implementation targets modeling and simulation of energy consumption and DVFS during workflow execution, whereas Chen and Deelman's focused in a realistic modeling of workflow management systems, including advanced scheduling techniques, such as task clustering and fault-tolerant scheduling, without emphasis on power-aware simulation.

The class diagram of the extension supporting power-aware DAGs in CloudSim is depicted in Fig. 3. A *WorkflowEngine* class, which models a system that manages the scheduling of tasks, has been developed. It reads DAG description files in

the DAX format, from the Pegasus project[1] and translates the DAG elements in CloudSim objects: tasks are translated to *Cloudlets* objects and files are converted to *DataItem* objects.

The WorkflowEngine schedules the DAG for execution. The DAG and the types and cost of available VM types are the input parameters for the DAG scheduling policy, which is defined as an abstract class *Policy*. Concrete implementations of this class incorporate algorithms for provisioning of VMs (i.e., decision on number of machines required by the application and their start and end times) and DAG scheduling.

At any moment when a VM is idle and the next task scheduled for execution on that VM is ready (i.e., all the required DataItems are already copied to the VM), the WorkflowEngine dispatches such task for execution. Upon completion of execution of a task, all the DataItems generated as output of the task are available in the VM where the task was executed. The WorkflowEngine then requests to the WorkflowDatacenter the transfer of DataItems to the VMs where children of the task are scheduled for execution.

WorkflowDatacenter class extends CloudSim's *Datacenter* class. The difference between the former and the latter is that the former includes new classes that help in modeling network contention for transfer of data between virtual machines, which are the *Channel* and *Transmission* classes. *Channel* represents a channel for transmission of data between VMs, whereas the *Transmission* class models one single data transfer.

The actual data (i.e., the files generated by one workflow task that is required by its children) is modeled in the *DataItem* class. At each simulation step, the amount of data of the *Transmission* that is considered already transferred is calculated based on the elapsed simulation time and the available bandwidth for the particular transmission. When the total amount of data defined in the object is complete, a message is sent to the WorkflowEngine advising the availability of the *DataItem*, so it can proceed with the dispatching of ready tasks.

5. DVFS Validation on one host

This section describes how the DVFS implementation in CloudSim has been validated using a simple test application. The test application has been created, like a benchmark, with different CPU load phases to be sure that the validation process handles different situations of frequency change.

The objective of this section is to validate the behavior of DVFS and model of energy consumption in CloudSim. The main idea is to execute a sequence of instructions on a real host with a Linux kernel with DVFS enabled and measure the total energy consumption. Then it comes to accurately simulate the same experiment in CloudSim with DVFS activated and compare both values of execution time and energy consumption between CloudSim and those of the real host in each mode of DVFS. This process needs to be very precise to have accurate results, this is why the methodology is especially emphasized in this article.

5.1. Experimental framework

All experiments were performed on a standard host (called *HOST* thereafter) equipped with an *Intel (R) Core (TM) 2 Quad CPU Q6700@2.66 GHz with 4 GB of RAM memory, running Ubuntu 10.4 LTS (Linux kernel 2.6.32)*. All measurements of energy consumption were made with a wireless *plogg*[2] power-meter which allows to measure and save energy consumption in real time. One value can be obtained each second with a very high precision ($>10^{-4}$ W).

5.2. Energy consumption calibration

In order to calibrate the calculation of the energy consumption in the simulator, it is first necessary to know the frequency values (Table 1) allowed by the CPU of the host. In CloudSim, the frequency is expressed in MIPS, indeed, simulator frequencies have been calculated proportionally to the host *HOST* frequencies values. Then, power values given by the host at 0% and 100% of CPU utilization, called $P_{CPUIdle}$ and $P_{CPUFull}$ for each frequency (i.e., $2 * Nb\_Freq$ measurements) are measured (Table 2) and used in the simulator to create the equivalent power model.

5.3. Methodology

The challenge is to perform an experiment that involves many frequency changes to test the behavior of DVFS, but also phases of constant load to use the power model over its entire range. In this scenario, the maximum CPU load value has been chosen to be 96% since it is enough to trigger frequency changes in dynamic DVFS modes.

Chronologically, the progress of the experiment, shown in Fig. 5, is:

- (*A*) progressive increase of CPU load from 0% to 96%.
- (*B*) a phase of stressed CPU load of 96%.

[1] https://www.confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[2] http://www.plogginternational.com/products.shtml.

Table 1
Available frequencies on *HOST* (in GHz), and those added in CloudSim (in MIPS).

Frequencies					
HOST (GHz)	1.60	1.867	2.133	2.40	2.67
$\%*F_{max}$	59.925	69.93	79.89	89.89	100
CloudSim (MIPS)	1498	1748	1997	2247	2500

Table 2
Powers (in W) given by *HOST* at 0% and 100% of CPU load for each frequency.

CPU load	Frequencies (GHz)				
	1.60	1.867	2.113	2.40	2.67
0% ($P_{CPUIdle}$)	82.7	82.85	82.95	83.10	83.25
100% ($P_{CPUFull}$)	88.77	92.00	95.5	99.45	103.0

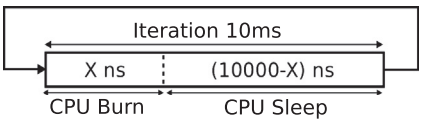


Fig. 4. Decomposition of loop in C language.

Table 3
Examples of CPU loads and execution durations of each [VM_x,$Cloudlet_x$] defined using $Host_1$ with a capacity of 1000 MIPS.

	VM/Cloudlet			
	C_1	C_2	C_3	C_4
CPU load (%)	10	10	2	2
Duration (s)	1.5	0.8	7.5	4

- (*C*) progressive decrease of CPU load from 96% to 50%.
- (*D*) peak CPU load to 80%.
- (*E*) progressive decrease of CPU load from 80% to 30%.
- (*F*) peak CPU load at 96%.
- (*G*) a phase of constant CPU load of 30%.

To run this scenario on the host *HOST*, a test application was implemented in C language. The desired average load is obtained by performing a loop, illustrated in Fig. 4 where each iteration is composed of a calculation period and a sleep period. When DVFS is enabled, the CPU load is checked every *sampling rate* (10 ms). To be sure that the decision of the DVFS *governor* is taken directly linked with the CPU load generated by the computation loop, each time iteration of the loop must be exactly equal to the interval *sampling rate*.

This scenario is modeled in CloudSim by running a set of *brokers*, each containing one or several VMs in which *cloudlets* are executed. As explained in Section 4.1, a *broker* may contain a *post-event*, which starts when all the VMs contained in it have finished their execution. By defining different types of VMs (with different MIPS) and different types of *cloudlets* (executing different number of instructions), it is possible to increase and manage the CPU load accurately. The size of a VM defines the percentage increase to the CPU load and the size of the *cloudlet* defines the execution time of the VM. So, this scenario is created by a set of *brokers* that launch a series of VMs and *cloudlet* pairs. Finally, to have the same interval *sampling rate* as in the Linux kernel, the simulation time interval is set to 10 ms.

Here is an example of how the CPU load is generated in CloudSim using different types of *Host/VM/Cloudlets*:

$Host_1$: 1000 MIPS.
VM_1: 100 MIPS, $Cloudlet_1$: 150 Millions of Instructions.
VM_2: 20 MIPS, $Cloudlet_2$: 80 Millions of Instructions.

Considering the above hosts, VMs, and *cloudlets*, four combinations of *VMs/Cloudlets* can be used to generate different CPU loads on $Host_1$, as follows:

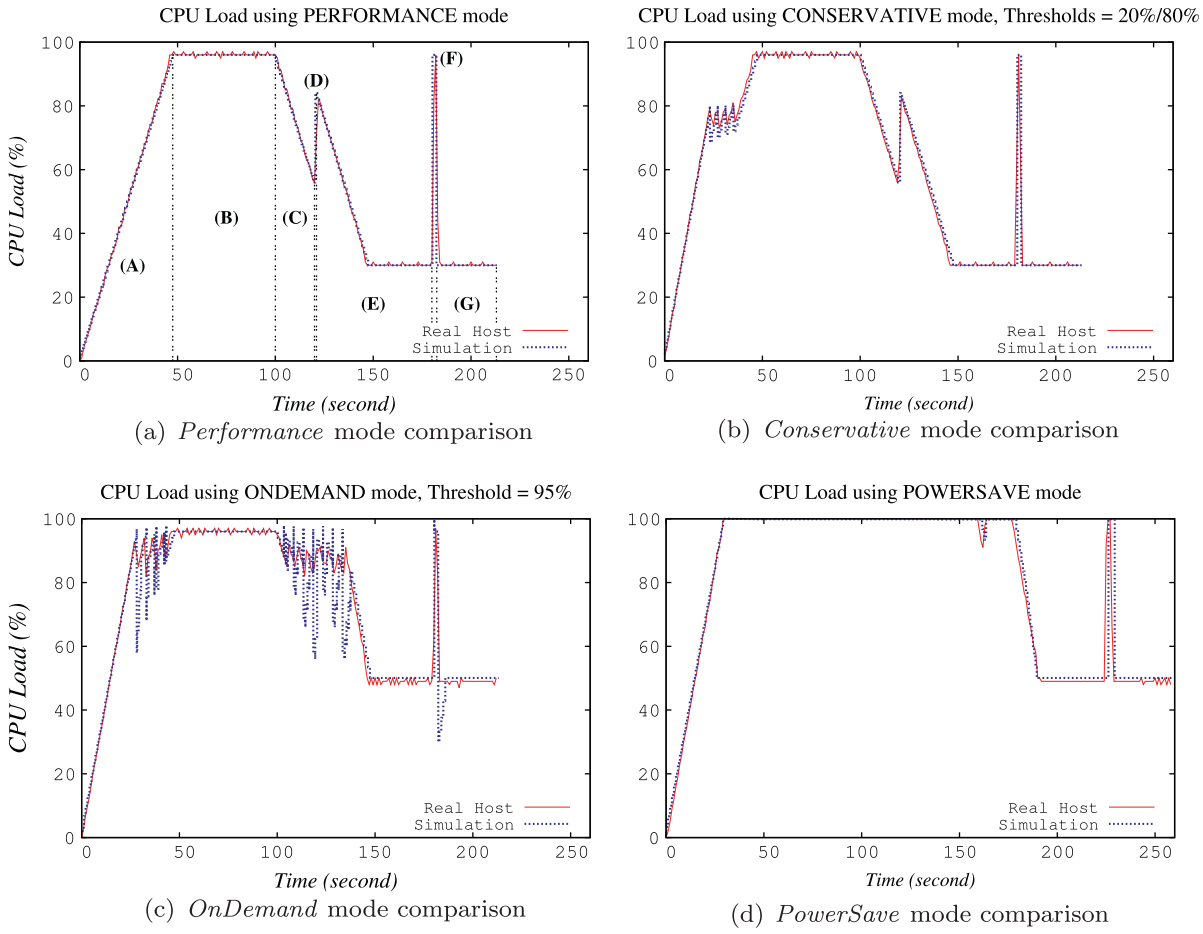


Fig. 5. CPU load comparison between the real host *HOST* and CloudSim simulation, using each DVFS mode.

$$C_1 = [VM_1, Cloudlet_1], \quad C_2 = [VM_1, Cloudlet_2]$$
$$C_3 = [VM_2, Cloudlet_1], \quad C_4 = [VM_2, Cloudlet_2]$$

Table 3 shows the CPU load and the length of each defined combination. In this example, if several couples of *VM/Cloudlet* like C_3 are launched one after the other, with an interval of 1 s between them, the CPU load increases smoothly (2%). If five couples of C_1 are launched at the same time, it allows the creation of a peak load of 50% during 1.5 s.

5.4. CPU load comparison

Fig. 5 presents CPU load curves at maximum frequency obtained during the experiment on the host *HOST* and during CloudSim simulation. At maximum frequency, the challenge was to have exactly the same CPU load, to be sure that all simulations started with the same conditions. Then the experiment was launched in the three other DVFS modes to evaluate the behavior of the simulator.

Scenario phases (*A*) to (*G*) are well distinguishable in Fig. 5. Indeed, *Performance* graph represents exactly the scenario defined in Section 5.3.

The *Conservative* CPU load graph (Fig. 5b) shows only small CPU load variations from phase (*A*) to (*B*) due to its progressive step by step increasing of the CPU frequency. No other changes can be observed in relation to the *Performance* graph, justified by the fact that CPU load never drops under its *down_threshold* (20%), implying that the CPU frequency stays at its maximum value until the end of phase (*G*).

The *OnDemand* mode (Fig. 5c) shows more CPU load variations during its execution. The behavior of the *OnDemand* mode favors host performance by increasing the CPU frequency to its maximum as soon as the CPU load exceeds the threshold. Once the CPU load frequency drops below this same threshold, the CPU frequency is decreased step by step. This interesting behavior can be easily observed, for example, during the (*C*) phase, in which abrupt CPU load changes can be observed: while the CPU load is being lowered, passing below the threshold of 95%, the *OnDemand governor* gradually decreases the CPU

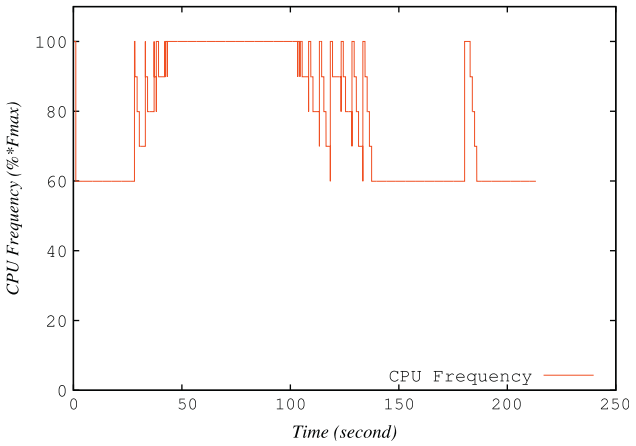


Fig. 6. CPU frequency changes during CloudSim simulations using the *OnDemand* mode.

Table 4
Results of the comparison between *HOST* and CloudSim in each evaluated DVFS mode. Percentages errors concern the energy consumption results.

DVFS mode	*HOST*		CloudSim		Error (%)
	Duration (s)	Energy (W h)	Duration (s)	Energy (W h)	
Performance	213	5.72	213	5.61	1.92
OnDemand	213	5.57	213	5.49	1.43
Conservative	213	5.68	213	5.64	1.71
PowerSave	259	6.37	260	6.33	0.63

frequency. Each time the CPU frequency is decreased, the CPU load increases proportionally until the frequency used begets a CPU load over 95%. At this moment, the *OnDemand governor* sets the CPU frequency at its maximum value and this situation occurs until the end of this phase. These CPU frequency changes can be observed in Fig. 6.

Finally, the analysis of the *PowerSave* mode (CPU frequency fixed at minimum) execution graph (Fig. 5d) allows to understand that this mode causes delay in tasks execution as the CPU is requested more computing power than it can provide. All scenario phases are sequentially executed, but by using the lowest CPU frequency, the CPU load never has an opportunity to decrease. The decreasing (*C*) phase is observable a short moment at *Time* $\simeq$ 165 s. The total execution time using this DVFS mode is significantly longer than the execution time obtained with the other modes.

5.5. Energy consumption comparison

Table 4 summarizes the execution time and energy consumption of various experiments and simulations, and compares the energy consumption value in terms of error rate.

It is interesting to note that in an experiment like the one presented in this section, results in terms of execution time and energy consumption follow the logic of the different modes of DVFS and there is a consistency between real measurements and the results of the simulator.

Conservative (Fig. 5b) and *OnDemand* (Fig. 5c) modes have execution times similar to the *Performance* for two different reasons. As explained in the previous section, the *Conservative* mode smoothly increases the CPU frequency to its maximum value during the (*A*) phase, and then continues using the fastest one because the CPU load never falls below its *down_threshold*. Indeed, from (*B*) to the end of the (*G*) phase, this mode has exactly the same behavior as *Performance* and there is no execution delay during the (*A*) phase, leading to an equal total execution time. About the *OnDemand* mode, it mainly prevents performance loss by increasing the CPU frequency to its maximum directly when necessary, so no execution delay occurs. In an energy consumption point of view, the *OnDemand* mode is the one that gives the best result because of its more responsive behavior to decrease the frequency compared to the *Conservative* mode. The result of the *PowerSave* mode can be explained because theoretically this mode gives the lowest live consumption at a given time *t*, but here the duration is much longer due to the low CPU frequency, which is not always favorable in terms of energy consumption for a CPU intensive application. Table 4 shows that the validation phase has been conducted in proper conditions, with accurate real measurements and well-defined simulations with a worst percentage energy consumption error of 1.92%.

The next section presents two DVFS use cases that involve more complex applications.

Table 5
Grid'5000 Reims site available frequencies, and power measurements at *Idle* and *Full* CPU states, which both use 0% and 100% of the whole 24 cores of a node, respectively.

Grid'5000 Reims site						
Available frequencies (GHz)		0.8	1.0	1.2	1.5	1.7
Power (W)	*Idle*	140	146	153	159	167
	Full	228	238	249	260	272

6. Evaluation

After validating the DVFS modeling in CloudSim with a simple application, this section presents two more complex DVFS use cases. The first one involves a parallel Grid application implementing an electromagnetic method called TLM (Transmission Line Matrix Method). The second one presents simulations of Cloud applications represented by DAGs which give attractive DVFS usages.

These two evaluation cases use applications that involve CPU-intensive computing, I/O and network communications. More precise power models for disk or network could be used, but it is not the main focus of this section and, as detailed by Da Costa et al. [27], the CPU power consumption takes the largest part of total power consumption of a host. However, the power consumption generated by disk and network usage has not been totally ignored in this section. Results given in Sections 6.1.6 and 6.2.2 are based on measurements performed on the Grid'5000 Reims site with disk and network stressed applications. Most of the time, results showed an increase of power consumption inferior to PDUs precision (6 W). In addition, these measurement inaccuracies lead to small error percentages on the energy consumption value, and are considered in both real experiments and simulations. The simulations power models used on the experiments are based on these real experimental values.

6.1. Parallel MPI application

6.1.1. TLM description

The Transmission Line Matrix method (TLM) is a numerical method for electromagnetic simulation that fills the environment of the electromagnetic field propagation with a network of transmission lines. The modeling of a propagation field inside a given medium is possible thanks to the equivalence that exists between the electric and magnetic fields and voltages and currents in a transmission line network. A detailed presentation of the TLM method, also in a three-dimensional approach, is presented by Hoeffer [28].

The resolution of this method in parallel on a Grid computing environment is based on a division of the structure along the three axes. Volumes are assimilated to tasks executed on each host, CPU, or processor core, exchanging information via the MPI library (Message Passing Interface). Indeed, the TLM task has to communicate (send data) with its neighbors, which introduces an important amount of time spent with network communication.

6.1.2. Application and hardware characteristics

The MPI library used for the TLM application case study is *OpenMPI*. One critical point of this library that has to be considered is the *pooling* time. The *pooling* time is the time during which a task is waiting to receive data sent by its neighbors. Although this could be regarded as a sleeping time, this is not the case when the *OpenMPI* library is used, because the CPU is fully used during this time. This is why it is very important, in energy-aware experiments, to take into account this parameter. A frequency change to a lower frequency during a *pooling* time is much less efficient than in a phase when the CPU load tends towards 0%. This characteristic of the MPI library and a *non-pooling* implementation are discussed by White and Bova [29].

For this first evaluation case, hosts of the Grid'5000 [30] Reims site have been used to run the real experiments. Power characteristics and available frequencies of this site are given in Table 5. These nodes have a surprising behavior using DVFS concerning the hard disk throughput. After many experiments using DVFS on different modes, either using a fixed frequency or a dynamic mode, it appeared that the time spent on I/O was also affected by the DVFS. In other words, if the CPU frequency decreases, so does the I/O speed. This means that on the architecture of these particular hosts[3], DVFS affects both the Front Side Bus frequency and the multiplier (to obtain the CPU frequency).

Knowing these characteristics, two models to study the energy consumption of this application can be defined. The first is the analytical one, which allows the computation of an estimation of the power consumption. The second one is the event model, which is used in the simulation, to represent the behavior of the application by a sequence of events. Results of these two models are analyzed and compared with real experiments values in Table 6.

[3] https://www.grid5000.fr/mediawiki/index.php/Reims:Hardware.

Table 6
Comparison between Grid'5000 Reims site experiment results obtained with the analytical model and simulations. Percentage errors are computed relatively to Grid'5000 experiments results. Experimental results are based on an average of 20 executions given a Time and Energy standard deviation value inferior to 150 s and 22 W h in both DVFS modes, respectively.

	DVFS modes							
	Performance				*OnDemand*			
	Time		Energy (W h)		Time (s)		Energy	
	Value	Error	Value	Error	Value	Error	Value	Error
Simulation	3790	−0.07%	478	−1.44%	4932	−0.1%	613	−0.8%
Analytic	3882	2.69%	499	2.88%	ø	ø	ø	ø
Experiments	3793	ø	485	ø	4937	ø	618	ø

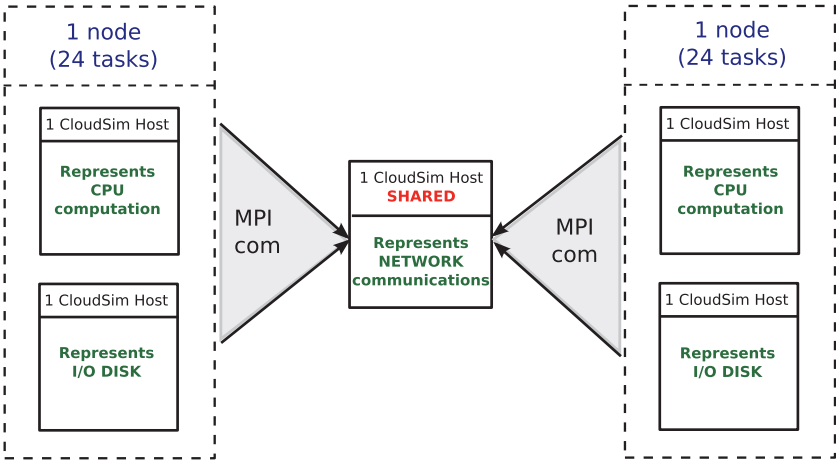


Fig. 7. Graphical representation of the functioning decomposition of the TLM in the simulator.

6.1.3. Analytical model

The model introduced next allows the calculation of the energy consumption of the experiments, if the frequency used is known. As explained below in Section 6.1.2, the model has to take into account the *pooling* time and the slow down relation between the CPU frequency and the disk throughput.

H: Node used.
T_{cpu} and T_{net}: CPU and Network time.
D_{disk}: Amount of data to be written on the disk in number of elements.
$Th_{disk}(f_i)$: Disk throughput at frequency f_i.
$F = \{f_1, f_2, \ldots, f_{n-1}, f_n\}$: Available frequencies on node H, with $f_{n-1} < f_n$.
$l^H_{cpu}(f_i)$ and $h^H_{cpu}(f_i)$: Power given by node H at 0% and 100% of CPU load.
$l^H_{disk}(f_i)$: Power given by node H when disk is used.

The prediction model of CPU, network and disk of the TLM presented by Alexandru et al. [31] was used: $T_{cpu} = C_1 + n_l n_x n_y n_z C_2$, $T_{net} = \left(L + \frac{n_x n_y}{D}\right) * 4n_l$ and $D_{disk} = n_x * n_y * n_z$, with n_x, n_y, n_z dimensions of the TLM environment, n_l the number of iterations, C_1 and C_2 constants estimated by past experiments, L and D are latency and network throughput.

The total energy E is:

$$E = T_{net} * h^H_{cpu}(f_i) + (T_{cpu} * f_n)\left[\frac{h^H_{cpu}(f_i)}{f_i}\right] + \left[\frac{D_{disk}}{Th_{disk}(f_i)}\right] * l^H_{disk}(f_i)$$

6.1.4. Event model

The event model represents how the TLM application has been modeled in the CloudSim simulator (Fig. 7). In the simulation, all 24 tasks are hosted by one CloudSim host with 24 cores. In one node, the generated amount of data I/O is written on the disk shared by the 24 tasks of a node. Therefore, in the simulation, the I/O throughput and capacity have to be represented by one host shared by all tasks. This method has been applied because the storage management of the simulator

does not allow slowing down the I/O speed when using DVFS. As explained in paragraph Section 6.1.2, the disk throughput is slowed down at the same rate than the CPU frequency on some architectures. Because of this, in the simulation, DVFS has to be enabled on the host that simulates the disk capacity. Finally, the network workflow generated by the MPI communication between tasks and nodes is represented by a single CloudSim host shared by all others. With this simulation architecture, all capacities of CPU, disks and Network are respected.

6.1.5. Experiments on Grid'5000

Experiments were conducted in the Grid'5000 Reims site, whose configuration is: *HP ProLiant DL165 G7, CPU: AMD Opteron 6164 HE 1.7 GHz, 12 MB L3 cache, 44 nodes with 2 CPUs to 12 cores per CPU (1056 cores)*.

The TLM was performed on both nodes using 100%, or about 48 cores, with one TLM task on each core. The input parameters are: $n_x = 172$, $n_y = 90$, $n_z = 12$ and $n_l = 26{,}000$. Throughput and network latency measured in Reims are: $D = 700$ Mbit/s and $L = 4 \times 10^{-5}$ s.

Deployment of TLM on Grid'5000 Reims site has been done in a *Kadeploy*[4] environment using two DVFS modes:

- Maximum Frequency (1.7 GHz) in *Performance* mode.
- *OnDemand* mode, which allows the kernel to dynamically change the CPU frequency depending on current CPU load.

During these experiments, a bash script collected and saved the power delivered by all used nodes. Grid'5000 Reims site allows the use of PDUs (Power Distribution Units), which give a power measurement every 3 s, with a precision of about 6 W.

6.1.6. Results

Table 6 shows the results between the two defined models defined real experiments. The analytical model allows the quick computation of an estimation of the execution time and energy consumed using equations related to the size of the TLM structure and power characteristics (CPU, network, and disk) of hosts. The event model is applied in the CloudSim simulator with a virtual representation of the application and hardware behavior as explained in Section 6.1.4. This model allows the simulation of the two DVFS modes, *Performance* and *OnDemand*, used in real experiments, and gives both accurate results of execution time and energy consumed. The analytical model gives also good results for the *Performance* mode, but unlike the event model, it cannot be used with the dynamic DVFS mode *OnDemand*. In fact, dynamical frequency changes regarding the CPU load on each host cannot be accurately modeled with this method.

About energy consumption values, the *OnDemand* mode achieves worse results than the *Performance* one. As explained in Section 6.1.2, this is caused by the slow down of the I/O time when the CPU frequency decreases and the *pooling* time during MPI communications. These results need to be more detailed because, in the standard DVFS behavior, the *OnDemand* mode always achieves better energy consumption results. On the Reims site, during a CPU load stressed phase, the *OnDemand* mode performs as expected, dynamically decreasing or increasing the frequency regarding its threshold. However, when a TLM task is waiting for data from its neighbors, the CPU load remains at 100% because of the *pooling* characteristic. In fact, the *OnDemand governor* does not change the CPU frequency and therefore introduces high power consumption. During an I/O phase, the CPU load goes to 0% and the *OnDemand governor* quickly decreases the CPU frequency, which leads to I/O slow down proportional to the new lower frequency. This behavior during network and I/O times is the reason why the *OnDemand* mode is not efficient in these experiments.

6.2. DAG simulations

The aim of the DAG simulations is to show the effectiveness of the use of DVFS, in different configurations, to save energy during DAG execution. An application represented by a DAG is an application with an execution flow split in ordered tasks (Fig. 9). One task can have both *children* that have to be executed after its end, and *parents* that have to be executed before its start. These dependencies are translated in files that are generated as a result of the execution of a parent and used as input for the execution of the child.

The longest path, from an entry task (i.e., tasks that do not have parents) to an exit task (i.e., tasks that do not have children) is called *critical path*. The minimum execution time is bounded by the duration of tasks in the *critical path* in the DAG. If a DAG contains several *critical paths*, a *critical sub-graph* which contains all the *critical paths* are defined and taken into account. Considering that these paths are not variable, the execution time of the DAG is equal to the end time of *critical path's* last task. Each task outside this *critical sub-graph* is called *non-critical* task and can be slowed down without risking delaying the DAG execution.

The workflow model considers tasks dependencies, execution time of each task, and the amount of data that each task has to send to its children. The time required for the data to be transmitted from the VM running the parent to the VM running the child determines the communication time of tasks, and this time is equal to 0 if both tasks are executed on the same VM. This communication time is taken into account during the analysis of the DAG to determine the *critical path*.

[4] Deployment system for cluster and grid computing: https://www.gforge.inria.fr/projects/kadeploy/.

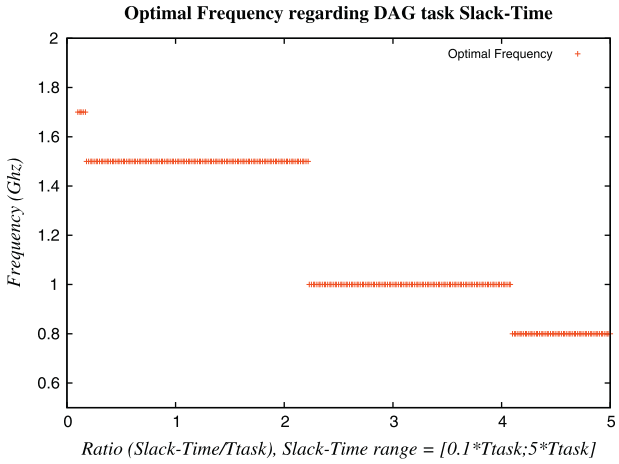


Fig. 8. Graph of optimal fixed frequency depending on Grid'5000 Reims site power characteristics and *Slack-Time*, which varied from $(0.1 * T_{task})$ to $(5 * T_{task})$. T_{task} is the execution time of a given task, and the X axis unit is $\frac{Slack-Time}{T_{task}}$.

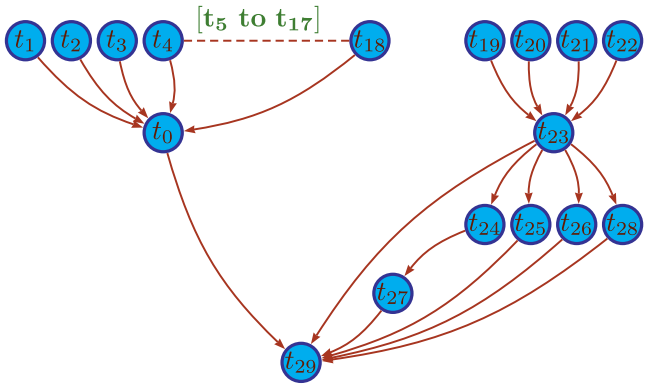


Fig. 9. The *Sipht* DAG application with 30 tasks used for simulations. For detailed informations about execution time and data item sizes of this DAG [5].

In order to establish a theoretical lower bound on the energy saving enabled by utilization of DVFS for DAG execution, the scheduling algorithm used for scheduling DAGs to VMs allocates each task to a different host. This forces data to be transferred at the end of each task execution, because tasks are not sharing virtual machines. Development of new scheduling algorithms able to optimize number of hosts and execution time are subject of future work.

The DAG application used for these simulations is *Sipht30*, which is depicted in Fig. 9. This DAG is an instance of *Sipht* which is a workflow to automate the encoding of genes. This workflow uses a fairly large number of applications but has a very stable behavior in both time execution and data sizes[5]. Data transfer times are inferior to the execution tasks times.

6.2.1. Slack-time algorithm and optimal frequency

The principle of slowing down *non-critical* task is to find the best *energy-aware* frequency to execute those tasks. This optimal frequency depends on the power characteristics of hosts used, the particular workflow topology, and the ratio between the task duration and the *slack-time* duration. The *slack-time* of a task is the time interval between its end and the earliest start time of its *child*. A task belonging to a *critical path* is not subject to *slack-time*.

The Grid'5000 Reims site's characteristics (showed in Table 5) have been used on this experiment. Optimal frequencies computed regarding the *slack-time* are shown in Fig. 8. This figure was obtained by computing the energy consumption of a task by varying its duration and its *slack-time*, for each frequency. Then, the lowest energy consumption value and its corresponding frequency, called optimal frequency, are saved for each ratio $\frac{Slack-Time}{T_{task}}$. In these experiments, the only valid values for optimal frequency are those available on the Grid'5000 Reims site, and this is why Fig. 8 shows discrete decrease frequency results.

[5] https://www.confluence.pegasus.isi.edu/display/pegasus/SIPHT+Characterization.

Table 7
Energy consumption results (in W h) comparison between the three DVFS modes used in DAG simulations. The gains of *UserSpace* and *OnDemand* modes are computed relatively to the power consumption value of the *Performance* mode.

DAG file*Sipht_30*	DVFS modes		
	Performance	*UserSpace* (F_{opt})	*OnDemand*
Energy (W h)	3241	2817	2751
Gain (%)	ø	13.1%	15.1%

The method for computing the *slack-time* for each *non-critical* task of the DAG was obtained from Kimura et al. [32], which presents a study of reducing energy consumption using *slack-time* and DVFS. Others studies about workflow overheads were presented by Chen and Deelman [33] and Nerieri et al. [34].

6.2.2. Results

The following Table 7 shows the energy consumption obtained of the Sipht30 application using different DVFS policies.

The *OnDemand* mode is very efficient: during CPU burn phases, the frequency is set to its maximum, and during lower CPU computation phases and after a task has finished its execution (i.e., CPU idle), the frequency decreases to its minimum available value, which has the lowest power consumption, and resulted in a gain of 15.1%. *Performance* and *UserSpace* (F_{opt}) modes do not change their frequency, which is of course not the most efficient behavior when the CPU is in idle state. Nevertheless, the use of the optimal frequency for each *non-critical* task is a suitable solution, resulting in a gain of 13.1% comparing to the *Performance* mode.

7. Conclusion

The purpose of this article was to provide an overview of the available energy-aware simulators and to describe the necessary tools required in a simulator to obtain accurate simulations in terms of energy consumption. After describing how both DVFS and DAGs have been incorporated in CloudSim, the methodology applied to evaluate the DVFS accuracy was explained in details in Section 5 and showed a worst energy consumption percentage error smaller than 2%, as compared to a real host, using a simple test bench application. This section highlighted the importance of the calibration phase, which demands a good knowledge of application, middleware, and hardware.

The evaluation section gives interesting conclusions about DVFS behavior, which was found to be closely linked with the internal architecture of the hosts and application functioning. Indeed, concerning the impact of the used architecture, experiments of the Grid application TLM demonstrated a strict relation between CPU frequency and I/O speed. In this section, one intrinsic application behavior that impacts DVFS was also explained, showing that the *pooling* time also points out the lack of efficiency of the *OnDemand* mode in this situation. In DAG simulations, the aim was to show the efficiency of slowing down *non-critical* tasks regarding their *slack-time* by using different DVFS modes. Results compared the energy consumption obtained using optimal fixed frequencies in *Performance* and *UserSpace* mode, and the dynamic *OnDemand* mode, which resulted in a gain of 15.1%. These two experimental situations demonstrated the importance of analyzing the environment in which DVFS is used, otherwise it may not necessarily always be efficient in terms of energy saving.

Proposed perspectives include establishing a set of energy-aware information about distributed platforms (such as Grid'5000) including their efficiency for different types of applications (parallel, DAGs, Client/Server, etc.). The next step is to explore the fact that the simulator is DVFS-enabled and energy-aware to test several energy saving strategies using different virtual machine scheduling policies and data center internal topologies. It will allow the comparison of existing strategies and their impact on energy, power, and performance. Another future step of this work is to use both energy-aware techniques and quality of service policies to develop green scheduling algorithms for different classes of applications.

Acknowledgement

The 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).

References

[1] M. Kocaoglu, D. Malak, O. Akan, Fundamentals of green communications and computing: modeling and simulation, Computer 45 (9) (2012) 40–46.
[2] B. Aksanli, J. Venkatesh, T. Rosing, Using datacenter simulation to evaluate green energy integration, Computer 45 (9) (2012) 56–64.
[3] M. Patterson, Energy Efficiency Metrics, Energy Efficient Thermal Management of Data Centers, Springer, 2012.
[4] S. Rivoire, P. Ranganathan, C. Kozyrakis, A comparison of high-level full-system power models, in: Proceedings of the 2008 Conference on Power Aware Computing and Systems, HotPower'08, USENIX Association, Berkeley, CA, USA, 2008, p. 3.
[5] R. Joseph, M. Martonosi, Run-time power estimation in high performance microprocessors, in: Proceedings of the 2001 International Symposium on Low Power Electronics and Design, ISLPED '01, ACM, New York, NY, USA, 2001, pp. 135–140.

[6] C. Isci, M. Martonosi, Runtime power monitoring in high-end processors: methodology and empirical data, in: Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 36, IEEE Computer Society, Washington, DC, USA, 2003, p. 93.
[7] G. Da Costa, H. Hlavacs, Methodology of measurement for energy consumption of applications, in: 11th IEEE/ACM International Conference on Grid Computing (GRID), 2010, IEEE, 2010, pp. 290–297.
[8] C.C. Keir, C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, Live migration of virtual machines, in: Proceedings of the 2nd ACM/USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2005, pp. 273–286.
[9] T. Kolpe, A. Zhai, S. Sapatnekar, Enabling improved power management in multicore processors through clustered dvfs, in: Design, Automation Test in Europe Conference Exhibition (DATE), 2011, pp. 1–6.
[10] H. Casanova, A. Legrand, M. Quinson, Simgrid: A generic framework for large-scale distributed experiments, in: Tenth International Conference on Computer Modeling and Simulation, 2008. UKSIM 2008, 2008, pp. 126–131.
[11] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, Groudsim: an event-based simulation framework for computational grids and clouds, in: Euro-Par 2010 – Parallel Processing Workshops, Lecture Notes in Computer Science, Springer, 2011.
[12] S. Bak, M. Krystek, K. Kurowski, A. Oleksiak, W. Piatek, J. Waglarz, Gssim-a tool for distributed computing experiments, Scientific Programming 19 (4) (2011) 231–251.
[13] K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz, Grid scheduling simulations with GSSIM, in: International Conference on Parallel and Distributed Systems, vol. 2, 2007, pp. 1–8.
[14] D. Kliazovich, P. Bouvry, Y. Audzevich, S. Khan, Greencloud: a packet-level simulator of energy-aware cloud computing data centers, in: IEEE Global Telecommunications Conference GLOBECOM, 2010, pp. 1–5.
[15] T. Issariyakul, E. Hossain, Introduction to Network Simulator NS2, Springer, 2011.
[16] R.N. Calheiros, R. Ranjan, A. Beloglazov, C.A.F. De Rose, R. Buyya, Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Software: Practice and Experience 41 (1) (2011) 23–50.
[17] Y.-K. Kwok, I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, ACM Computing Surveys 3 (4) (1999) 406–471.
[18] Z. Shi, J.J. Dongarra, Scheduling workflow applications on processors with different capabilities, Future Generation Computer Systems 22 (6) (2006) 665–675.
[19] J. Yu, R. Buyya, K. Ramamohanarao, Workflow scheduling algorithms for grid computing, in: F. Xhafa, A. Abraham (Eds.), Metaheuristics for Scheduling in Distributed Computing Environments, Springer, 2008.
[20] A. Hirales-Carbajal, A. Tchernykh, R. Yahyapour, J.L. González-Garćıa, T. Röblitz, J.M. Ramírez-Alcaraz, Multiple workflow scheduling strategies with user run time estimates on a grid, Journal of Grid Computing 10 (2) (2012) 325–346.
[21] E.-K. Byun, Y.-S. Kee, J.-S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (8) (2011) 1011–1026.
[22] S. Abrishami, M. Naghibzadeh, D. Epema, Deadline-constrained workflow scheduling algorithms for IaaS clouds, Future Generation Computer Systems 29 (1) (2013) 158–169.
[23] H. Kimura, M. Sato, Y. Hotta, T. Boku, D. Takahashi, Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster, in: Proceedings of the 2006 IEEE International Conference on Cluster Computing, IEEE Computer Society, 2006.
[24] L. Wang, G. von Laszewski, J. Dayal, F. Wang, Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with dvfs, in: Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, IEEE Computer Society, 2010, pp. 368–377.
[25] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, K. Vahi, Characterization of scientific workflows, in: Proceedings of the 3rd Workshop on Workflows in Support of Large-Scale Science (WORKS'08), IEEE, 2008.
[26] W. Chen, E. Deelman, WorkflowSim: A toolkit for simulating scientific workflows in distributed environments, in: Proceedings of the 8th IEEE International Conference on eScience (eScience'12), IEEE Computer Society, 2012.
[27] G. Da Costa, H. Hlavacs, K. Hummel, J.-M. Pierson, Modeling the energy consumption of distributed applications, in: I. Ahmad, S. Ranka (Eds.), Handbook of Energy-Aware and Green Computing, Chapman & Hall, CRC Press, 2012 (chapter 29).
[28] W. Hoeffer, The transmission-line matrix method – theory and applications, IEEE Transactions on Microwave Theory and Techniques 33 (10) (1985) 882–893.
[29] J. White III, S. Bova, Wheres the overlap? An analysis of popular MPI implementations, in: Proceedings of the Third MPI Developers and Users Conference, Citeseer, 1999.
[30] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quetier, O. Richard, Grid'5000: a large scale and highly reconfigurable grid experimental testbed, in: The 6th IEEE/ACM International Workshop on Grid Computing, 2005, p. 8.
[31] M. Alexandru, T. Monteil, P. Lorenz, F. Coccetti, H. Aubert, Efficient large electromagnetic problem solving by hybrid tlm and modal approach on grid computing, in: International Microwave Symposium, Montral, Canada, 17–22 June 2012.
[32] H. Kimura, M. Sato, Y. Hotta, T. Boku, D. Takahashi, Emprical study on reducing energy of parallel programs using slack reclamation by dvfs in a power-scalable high performance cluster, in: IEEE International Conference on Cluster Computing, IEEE, 2006, pp. 1–10.
[33] W. Chen, E. Deelman, Workflow overhead analysis and optimizations, in: Proceedings of the 6th Workshop on Workflows in Support of Large-Scale Science, WORKS '11, ACM, New York, NY, USA, 2011, pp. 11–20.
[34] F. Nerieri, R. Prodan, T. Fahringer, H. Truong, Overhead analysis of grid workflow applications, in: Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, IEEE Computer Society, 2006, pp. 17–24.