A Simulator to Assess Energy Saving Strategies and Policies in HPC Workloads

Manuel F. Dolz Juan C. Fernández Sergio Iserte Rafael Mayo
Enrique S. Quintana-Ortí

Dpto. de Ingeniería y Ciencia de los Computadores,
Universitat Jaume I, 12.071–Castellón, Spain
{dolzm,jfernand,siserte,mayo,quintana}@uji.es

ABSTRACT

In recent years power consumption of high performance computing (HPC) clusters has become a growing problem due, e.g., to the economic cost of electricity, the emission of carbon dioxide (with negative impact on the environment), and the generation of heat (which reduces hardware reliability). In past work, we developed *EnergySaving cluster*, a software package that regulates the number of active nodes in an HPC facility to match the users' demands. In this paper, we extend this work by presenting a simulator for this tool that allows the evaluation and analysis of the benefits of applying different energy-saving strategies and policies, under realistic workloads, to different cluster configurations.

1. INTRODUCTION

HPC clusters have been widely adopted by companies and research institutions for their data processing and scientific computing centers because of their parallel performance, high scalability, and low acquisition cost. However, further deployment of HPC clusters is limited due to their considerable economic costs in terms of energy consumption, required both by the computing hardware and the cooling backup. Furthermore, the electricity cost is not the only problem; in general, energy consumption results in carbon dioxide emission, a hazard for the environment and public health, and heat, which reduces reliability of hardware components [11]. The situation will become worse in the future: a look to the Green500 list from June 2011 [1] indicates that, as-of-today, the most power-efficient supercomputer (NNSA/SC Blue Gene/Q Prototype 2) delivers 2,097.19 MFLOPS/W. A simple calculation thus reveals that attaining the EXAFLOPS arithmetic rate (10^{18} floating-point arithmetic operations, or flops, per second) with the current technology would require 476.83 MW. That amounts roughly for 50% of the electrical production capacity of a modern nuclear reactor! Although the EXAFLOPS challenge will undoubtedly render groundbreaking scientific discoveries, it is also certain that it asks for greener hardware technology as well as more efficient system software, middleware, and application algorithms from the energy viewpoint [4].

During the last decade an increasing number of efforts have been conducted in order to conserve energy on distributed systems. Here we concentrate on energy-aware computing and review the state-of-the-art on energy conservation policies and algorithms. The work done on energy-aware computing can be divided into two levels: node and cluster. At the node-level, some algorithms leverage DVS (Dynamic Voltage Scaling) technology in the context of mobile devices, commercial web servers and scientific HPC systems. Other node-level mechanisms include Intel Turbo Boost [2], Core on/off and Request Batching [15]. However, although node-level approaches (either in hardware or in software) have been shown to preserve a significant fraction of the total energy consumption, power consumption of an idle server is still relatively high (e.g., 190W for an IBM eServer 325 [18]). Thus, in order to increase the energy savings, existing node-level approaches can be enhanced with a higher level energy-aware management layer (cluster-level). Such approaches can address the problem of high idle power consumption either by consolidating individual tasks or virtual machines on a subset of nodes while switching off unused resources.

In the cluster-level context a well-known energy management technique is DVFS (Dynamic Voltage and Frequency Scaling). This mechanism entails reducing the system energy consumption by simultaneously decreasing the CPU supply voltage and the clock frequency (CPU speed). DVFS has been leveraged by a large number of works with the purpose of reducing energy consumption [12, 7, 5]. The authors in [8] present an energy-aware method that partitions the workload and reduces energy consumption in multiprocessor systems with support for DVS. Freeh et al. [16] analyze the energy-time trade-off of a wide range of applications running on HPC clusters. In [10], the authors use economic criteria to dispatch jobs to a small set of active servers, while other servers are transitioned to a low energy state.

Alternative strategies to limit power consumption and the associated cooling of HPC clusters are based on switching on/shutting down nodes according to the demands from the users' applications. An algorithm that aims at balancing the workload depending on both the total load imposed on the cluster and the power and performance implications of turning nodes off is described in [19]. Several policies to combine DVFS and turning on/off nodes to reduce the aggregate power consumption of a server cluster during periods of reduced workload are presented in [14]. We have developed a tool, EnergySaving Cluster (ESC) [13], that employs a number of rules to activate/deactivate nodes for a full adaption of the system behavior to the computational power requirements. This tool has been designed and implemented as a module (Roll) for Rocks® [17] and employs the Sun® Grid Engine (SGE) [20].

In this paper we present the ESC Simulator (ESCS) based in our ESC tool. This simulator reproduces the activity of this tool under realistic workloads and for various platform configurations. The simulator applies the desired activation/deactivation policies to a given input workload, and can be easily configured to accommodate different platform configurations. Among other statistics, the simulator reports the percentage of the time that each node in the cluster is turned on/off and, therefore, offers an estimation of the energy consumption. Therefore, the main benefit of this simulator is that it can serve as a demonstrator/testing tool of the energy savings that a particular power-aware policy can deliver compared with a conventional cluster in which all nodes are permanently active.

The rest of the paper is organized as follows: Section 2 describes the ESCS. In Section 3, we offer results from several simulations with different workloads that show the benefits of the tool. Finally, Section 4 summarizes the conclusions.

2. DESCRIPTION OF THE SIMULATOR

In this section, we first introduce a model for the energy consumption of a node and then describe the ESCS.

2.1 Model of the node energy consumption

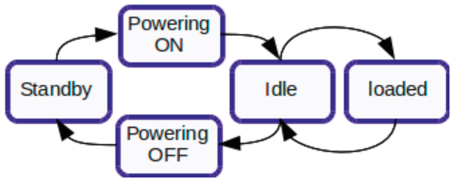


Figure 1: Model for the power states of a cluster node.

In order to obtain the energy consumption of a cluster we need a model of the energy consumption of the system nodes. In the current version of the simulator, we use a simple model consisting in 5 states, shown in Figure 1. Each vertex in the figure corresponds to a different state in the model:

- **Standby**. The node is off but still consumes a residual amount of energy due to the hardware that remains active waiting for a switch on command. The only parameter that characterizes this state is the power consumption.
- **Powering on**. Once a node has been selected to be powered on, a signal is sent to it. From this moment until the node is ready to execute a job there is a certain consumption of energy. The model in this state includes both the energy consumption and the time, from the instant the switch on signal is received to the moment in which the node is able to execute its first job.
- **Powering off**. Powering down a node requires a certain period from the moment the node receives the request till it reaches the standby state. The model needs the energy consumption and the time required to evolve to the standby state.
- **Idle**. In this state, the node is powered on but has no job assigned. The model only considers the power consumption of the idle state.
- **Loaded**. In this state, the node is powered on and executing a job. In the current version of the simulator, we assume that a job running in a node always employs its full computational power.

The simulator includes 7 major modules, illustrated in Figure 2: workload file loader, system configuration, queueing scheduler, energy saving scheduler, simulation, statistics, and website interface.

We have applied a modular design using Python to accommodate flexibility in the main functions of the simulator (control of the queueing system, statistics gathering, and application of node activation/deactivation policies). The tool uses a B-Tree to perform simulations based on temporal scheduling of events. It recognizes eleven events, which will be described later.

2.2 Workload file loader module

This module receives a workload file in *standard workload format* [9] as input. This file has two parts:

- The first four lines define the global aspects of the workload file: the total number of jobs of this file, the log start date, the log finish date, the number of nodes in the cluster, the number of processors in the cluster, and the number of queues in the system.
- The remaining lines describe the jobs running in the cluster. For each job, the file stores its identifier, the submission time, the user responsible of the job, the queue to which it belongs, the number of processors used, and the duration of the job.

From this workload file, this module builds a B-Tree containing all jobs in chronological order. The result of the execution of this module is a B-Tree consisting of events of type *a new job is submitted to the system* (one of the eleven types of events).

2.3 System configuration module

This module receives as input a configuration file, in *cfg* format, with the following information:

- Users of the system, the groups that they belong to, and the configuration of the queues for each user group.

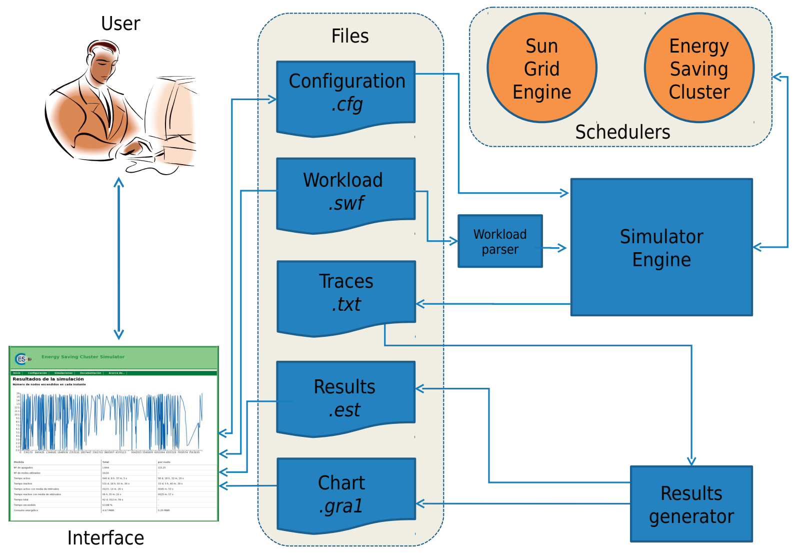


Figure 2: Modules of the ESCS.

- Nodes in the cluster and parameters of each group of nodes in the cluster.
- General operations of the simulator: parameters defining the policies applied to job executions, energy saving policies, and the duration of the various events occurring during the simulation.

From this configuration file, this module builds different Python structures. The most important one is the *Python dictionary*, which will be used by all other simulator modules except the workload file loader. In particular, the following Python dictionaries are created: nodes, queues, users, idle nodes, jobs, and groups of users.

2.4 Queueing system scheduler

The simulator employs a scheduler similar to the Sun Grid Engine (SGE) queue system to handle the execution of jobs. For each existing queue, the FIFO policy determines the schedule of the queued jobs. This module uses nodes and queues Python dictionaries. Due to the modular structure of the simulator, adding new policies to it, like those provided by SGE, is extremely easy.

2.5 Energy saving scheduler

This module is shared with that of the ESC, but employs the interfaces provided by the queuing system scheduler module to check and query the status of jobs, as well as activate/deactivate nodes. This module uses the following Python dictionaries: nodes, queues, idle nodes, users, and groups of users.

The module simulates the activation/deactivation of nodes according to the needs of the queue system's user. It compares the threshold parameters set by the system configuration module and the current values of these parameters from the queueing system scheduler to check if any of the activation/deactivation conditions is satisfied.

The threshold conditions are not necessarily equal for all users as it is possible to create multiple user groups with different values for the threshold parameters.

2.5.1 Node activation conditions

Nodes can be turned on if any of the following conditions holds:

- *There are not enough appropriate active resources to run a job.* That is, as soon as the system detects that a job does not have enough resources because all the nodes that contain the appropriate type of resource are powered down, nodes are turned on to serve the request.
- *The average waiting time of an enqueued job exceeds a given threshold.* The system configuration module defines an upper bound on the average waiting time in queue for the jobs of each group. When the average waiting time of an enqueued job exceeds the threshold assigned to the corresponding user's group, the system turns on nodes which contain resources of the same type as those usually requested by the same user.
- *The number of enqueued jobs for a user exceeds the maximum value for its group.* In this case, this module selects and switches on nodes which feature the properties required by most of the enqueued jobs.

In the ESC package there are several options to select the (candidate) nodes that will be activated. Currently, the simulator only implements an ordered policy where the list of candidate nodes is sorted in alphabetical order using the name of the node (hostname).

2.5.2 Node deactivation conditions

The following parameters define the conditions to turn off a node:

- *The time that a node has been idle.* exceeds an a threshold set by the system configuration module.
- *The average time waiting for users' jobs is less than a threshold set by the administrator.* The system configuration module defines a lower bound for the queue waiting time of the jobs of each group of users. In case the average waiting time of a user's job is lower than the threshold assigned to its group, this module turns off a node (choosing one based on a least-recently-used policy).

2.6 Simulation module

The simulation module looks up the B-Tree for the next event in time, analyzes it, and adopts the necessary actions to process this event, repeating this loop for all jobs of the workload. This module continually inserts events in the B-Tree like for example:

- Queue scheduler and energy saving scheduler are periodically launched. The simulator records the starting events associated with these two schedulers.
- Every time that an event for a job startup, a node turn on, or a node turn occurs, the correspondent ending events are recorded.

There are 11 events that may appear during the execution of the simulation:

- *Node turn-on starts*: It changes the state of a node to "powering on".
- *Node turn-on ends*: It changes the state of a node to "idle", and updates the Python dictionary of idle nodes with the instant of time when this event happened.
- *Node turn-off starts*: It changes the state of a node to "powering off".
- *Node turn-off ends*: It changes the state of a node to "standby".
- *New job is submitted to the system*: The job is added to the corresponding job queue and the time of the event is annotated in the Python dictionary of the job.
- *Job execution starts*: It changes the state of a node to "loaded", and saves the waiting time from the submit time to the start time of the execution of this job in the Python dictionary of the user. This waiting time is updated periodically as it is used by the energy saving scheduler.
- *Job execution ends*: This event produces the following actions:
  - It increases the counter of finished jobs.
  - The job is deleted from the queue to which it belonged.
  - For each node involved in the execution of this job, it calculates its new state taking into account the number of free cores in this node; the state of the node is updated with this information.
  - Finally, the total execution time is computed and saved.
- *Energy saving scheduler starts*: This event is added according to the energy saving scheduling periodical time. Also the event of *energy saving scheduler ends* is added in the B-Tree, taking into account the duration of the event.
- *Energy saving scheduler ends*: The action to perform depends on the decision taken by the energy saving scheduler. There are two possibilities: Turn on or turn off nodes. The appropriate events of *node turn-on/off starts* are recored in the B-Tree. It also records events associated with *node turn-on/off ends*.
- *Queue system scheduler starts*: This event is added according to the queue scheduling periodical time. The event of *queue system scheduler ends* is also added to the B-Tree.
- *Queue system scheduler ends*: The queueing scheduler decides which is the job to run. For each node involved in the decision of the queueing scheduler, the event of *job execution starts* is added in the B-Tree. It also attaches events of *job execution ends*.

The simulation module produces a trace file for each simulation. For each event processed, this module saves a line in this file containing several data: the time when each event occurs, elements involved, and decisions to take. The trace file is therefore the result of any decisions taken during the execution of the simulation. This file is needed to compile statistics on the simulation.

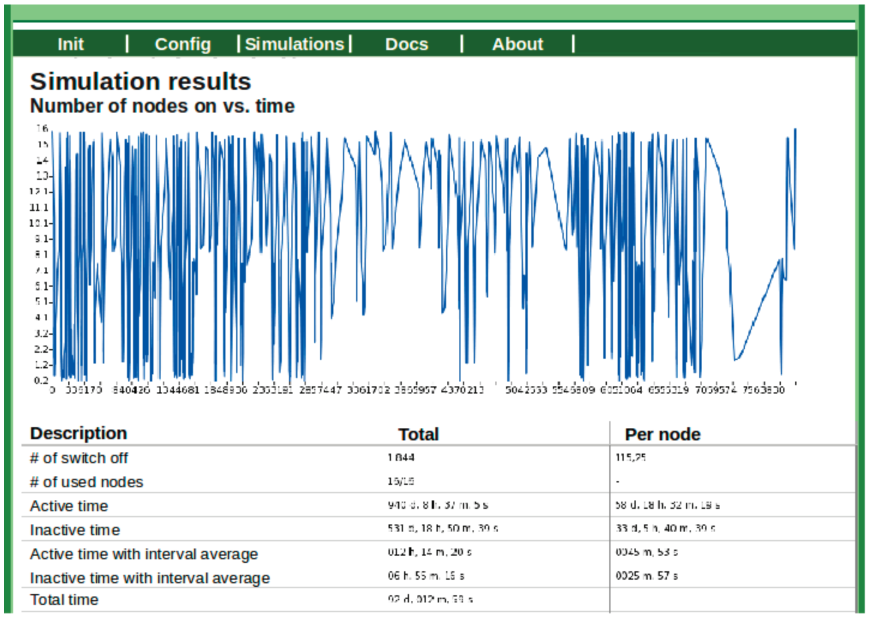
2.7 Statistics module

When the simulation finishes, all the submit, start and finish times of jobs and activation/deactivation transitions of nodes, with their specific times, are stored in the database. This application gathers several statistics, such as total active/inactive time per node, job latency, and so on; and prepares tables, graphs and statistics to evaluate energy consumption. In particular, it calculates the following data: maximum number of active nodes, number of shutdowns during the simulation period, average queue waiting time, average queue execution time, average user waiting time, average user execution time, average node execution time, active time, idle time, and total time of the simulation. From these values, it is possible to elaborate graphs illustrating, e.g., the number of active nodes at any moment in time.

This module also stores for each core the changes of states occurring during the simulation. The goal is to graphically represent what happened in each core during the simulation using Paraver [6], a flexible performance visualization and analysis tool. For this purpose, this module generates a file, in *prv* format, which contains the information that Paraver needs to display the results of the simulation. There is also a configuration file, in *pcf* format, to configure the states of the cores and the colors employed in the Paraver graphical representation.

2.8 Website interface

The website provides complete control over the simulator. The contents of the website are divided into pages, with the following structure:



Simulation results

Number of nodes on vs. time

Description	Total	Per node
# of switch off	1 844	115,25
# of used nodes	16/16	-
Active time	940 d, 8 h, 37 m, 5 s	58 d, 18 h, 32 m, 19 s
Inactive time	531 d, 18 h, 50 m, 30 s	33 d, 5 h, 40 m, 30 s
Active time with interval average	012 h, 14 m, 20 s	0045 m, 53 s
Inactive time with interval average	06 h, 55 m, 16 s	0025 m, 57 s
Total time	92 d, 012 m, 59 s	

Figure 3: Website page with the results of a simulation.

- Home: Welcome Page.
- Setting: To configure and launch executions.
- Simulations: View of the simulations (in progress or completed).
- Documentation: Manuals or other documents for the "ESCS."
- About...: Information on the web site.

The operations that can be performed from the website of the simulator include:

- Set the parameters of a simulation.
- Import configuration files to apply them to a simulation.
- Import workload files for simulation.
- Perform concurrent simulations and check the progress of their execution.
- Perform management operations on the simulations, such as:
  - Abort simulation.
  - Clear results of a simulation.
  - View errors during a simulation, if any.
  - View simulation results, composed by a line graph and a table with all the information obtained by the simulator; see Figure 3.

3. EXPERIMENTAL RESULTS

We have used a pair of synthetic workloads from the ParallelWorkloads Archive [3] to test our simulator: NASA Ames iPSC/860, consisting of 42,264 jobs, and OSC Linux Cluster, composed of 80,714 jobs.

Table 1: Information of the simulation of NASA and OSC workloads.

Workload	Time (d,h,m,s)	MWh
NASA without ESC	92 d,0 h,3 m,43 s	52.93 MWh
NASA with ESC	92 d,0 h,12 m,59 s	38.73 MWh
OSC without ESC	677 d,2 h,55 m,51 s	168.60 MWh
OSC with ESC	997 d,9 h,9 m,56 s	117.65 MWh

To obtain the results from the application of our simulator to these two benchmarks we have determined the following parameters:

- Time for a shutdown: 480 s.
- Time for a power on: 555 s.
- Energy consumption in "powering off" state: 10.79 Wh.
- Energy consumption in "powering on" state: 13.71 Wh.
- Power dissipation in "standby" state: 2 W.
- Power dissipation in "idle" state: 150 W.
- Power dissipation in "loaded" state: 230 W.

The results for these two benchmarks are reported in Table 1. The information there reveals that it is possible to attain important energy savings using the simple power on/off policy implemented in ESC. For the OSC workload, the energy consumption with ESC is 0.697× that obtained without using it. However, the use of ESC increases the time needed to process all the jobs by a factor of 1.47. On the other hand, with the NASA workload, the time is increased only by 1.000069, i.e. there is no penalty due to ESC, but the energy consumption of this alternative is only 0.73× that of the energy consumption without ESC.

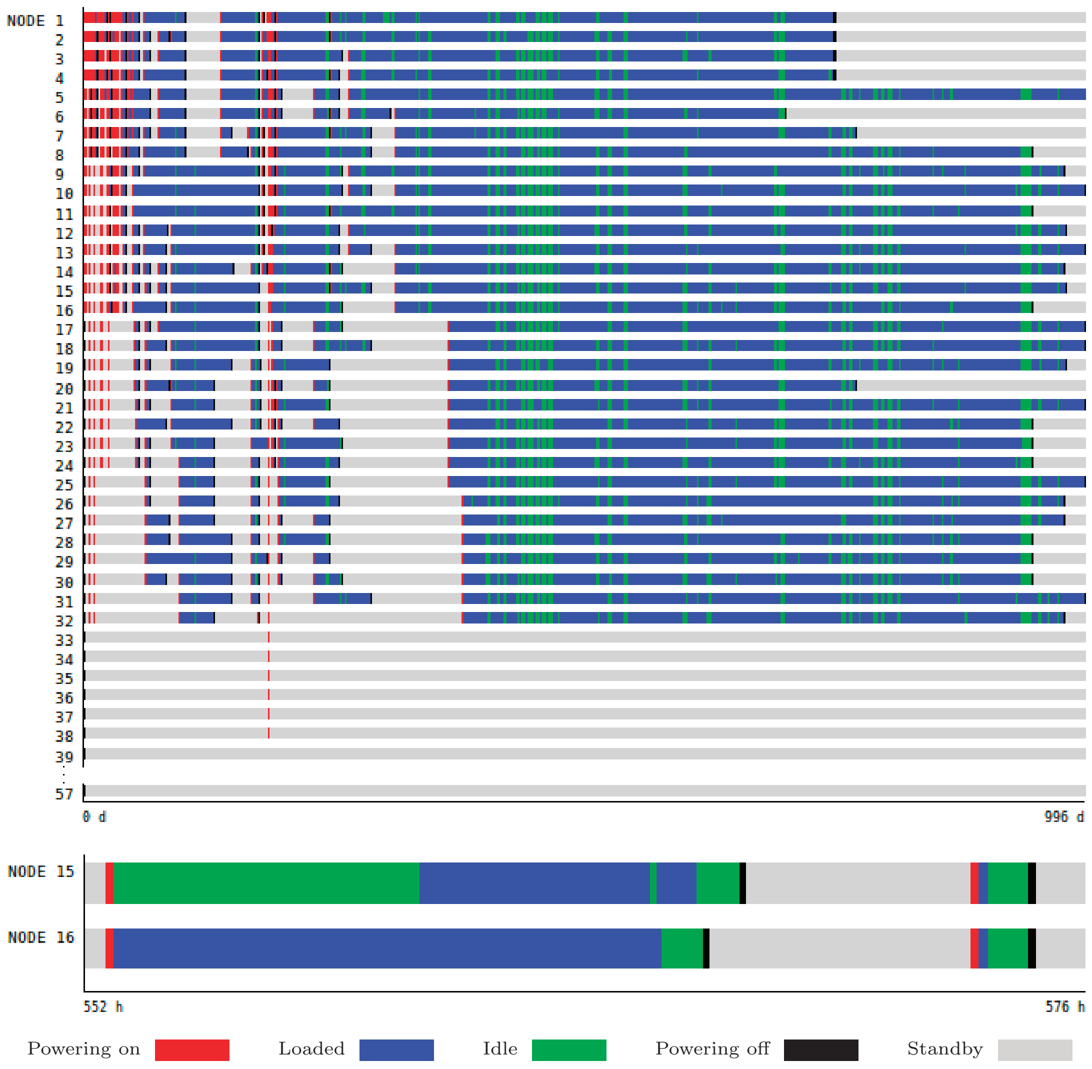


Figure 4: Full and detailed graphical representation for OSC workload using Paraver (top and bottom, respectively).

Table 2: Detailed results for OSC workload with ESC.

Measure	Total	Per node
Number of shutdowns	750	13,15
Maximum active nodes	32 of 57	-
Active time	24,010d, 19h, 27m, 52s	421d, 5h, 48m, 54s
Inactive time	32,782d, 22h, 58m, 20s	575d, 3h, 21m, 1s
Active time with average of active intervals per node	34d, 14h, 20m, 42s	14h, 34m, 2s
Inactive time with average of inactive intervals per node	47d, 5h, 42m, 16s	19h, 53m, 22s

Table 2 reports more detailed results for the OSC workload with ESCS. Specifically, the first row in the table shows the number of node shutdowns during 997 days: 750 shutdowns for a cluster with 57 nodes is a reduced number. This means that, in average, a node was activated/deactivated slightly more than 13 times in 997 days. The following rows show the active/inactive average time for all nodes in the cluster. To obtain these figures, we collected the active/inactive time per node to calculate the total average. This metric illustrates the time that the nodes are powered on: basically 421 days of a total of 997 days, or 42% of the time. The last two rows of results in the table display the total active/inactive average time of nodes from the local averages of active and inactive intervals per node, in this case the inactive time with average of inactive intervals is 20 hours.

Figure 4 shows the changes of states during the simulation of OSC workload with ESC using Paraver. The different colors identify the possible states of nodes: powering on, loaded, idle, powering off and standby. The top part of the figure offers the complete trace for the OSC workload while the bottom one zooms the activity of nodes 15 and 16 during a period of 24 hours.

4. CONCLUDING REMARKS

We have developed a simulator to evaluate the impact of energy saving policies in an HPC cluster. The tool is highly efficient: Simulation of months of execution in a cluster are reduced to minutes which accelerates the analysis of the data. In addition, the simulator provides data that eases the evaluation of the power performance of a system by testing different solutions and approaches in a matter of a few minutes. The modular design of the simulator enhances its flexibility, so that adding new features to the simulator is relatively easy.

Acknowledgments

This work was supported by project CICYT TIN2011-23283 of the MINECO and FEDER.

5. REFERENCES

[1] The Green500 list - june 2010. http://www.green500.org/.

[2] Intel2008: Intel turbo boost technology in intel core microarchitecture (Nehalem) based processors. November 2008.

[3] Logs of real parallel workloads from production systems. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[4] S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53:86–96, 2010.

[5] T. Bletsch, V. Freeh, D. Lownenthal, B. Rountree, M. Schulz, and B. de Supinski. Adagio: making DVS practical for complex HPC applications export. *Proceedings of the 23rd international conference on Supercomputing (ICS '09)*, pages 460–469, 2009.

[6] BSC. *Paraver*. http://www.bsc.es/computer-sciences/performance-tools/paraver.

[7] K. W. Cameron, R. Ge, and X. Feng. Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters. *Conference on High Performance Networking and Computing Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 34, 2005.

[8] X. Changjiu, L. Yung-Hsiang, and L. Zhiyuan. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. *Proc. 44th Annual Conf. Design Automation, San Diego, CA, USA, ACM*, pages 664–669, June 2007.

[9] S. J. Chapin, W. Cirne, D. G. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. *In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (Eds.), Springer-Verlag, Lect. Notes Comput. Sci.*, 1659:66–89, 1999.

[10] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centres. *Proc. 18th ACM Symp. Operating System Principles, Banff, Canada*, pages 103–116, 2001.

[11] W. chun Feng, X. Feng, and R. Ce. Green supercomputing comes of age. *IT Professional*, 10:17–23, 2008.

[12] R. Dick, H. Yang, L. Shang, Y. Liuand, and H. Wang. Thermal vs energy optimization for DVFS-enabled processors in embedded systems. *International 8th Symposium on Quality Electronic Design (ISQED '07)*, pages 204–209, March 2007.

[13] M. Dolz, J. C. Fernández, R. Mayo, and E. S. Quintana-Ortí. Energysaving cluster roll: Power saving system for clusters. *Architecture of Computing Systems - ARCS 2010, Lecture Notes in Computer Science (LNCS)*, 5974:162–173, 2010.

[14] E. Elnozahy, M. Kistler, and R. Rajamony. Energy-efficient server clusters. *Workshop on Mobile Computing Systems and Applications*, February 2002.

[15] E. Elnozahy, M. Kistler, and R. Rajamony. Energy conservation policies for web servers. *Proc. of the 4th USENIX Sympsium on Internet technologies*, 2003.

[16] V. Freeh, F. Pan, D. Lowenthal, N. Kappiah, R. Springer, B. Rountree, and M. Femal. Analyzing the energy-time tradeoff in high-performance computing applications. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):835–848, June 2007.

[17] U. of California. *Rocks® Clusters*. www.rocksclusters.org.

[18] A. Orgerie, L. Lefevre, and J. Gelas. Chasing gaps

between bursts: Towards energy efficient large scale experimental grids. *The 9th International Conference on Parallel and Distributed Computing. Applications and Technologies (PDCAT '08)*, pages 381–389, December 2008.

[19] E. Pinheiro, R. Bianchini, E. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. *Workshop on Compilers and Operating Systems for Low Power, September*, 2001.

[20] I. Sun Microsystems. *Sun® Grid Engine*. `http://gridengine.sunsource.net`.