TOWARDS ENERGY AWARE RESERVATION INFRASTRUCTURE FOR LARGE-SCALE EXPERIMENTAL DISTRIBUTED SYSTEMS

LAURENT LEFEVRE and ANNE-CECILE ORGERIE
INRIA, Université de Lyon, Ecole Normale Supérieure de Lyon
LIP (CNRS, ENS Lyon, UCB Lyon, INRIA)
46 Allée d'Italie, 69364 Lyon Cedex 07, France
laurent.lefevre@inria.fr - annececile.orgerie@ens-lyon.fr

December 11, 2019

Abstract

While an extensive set of research projects deal with the issue of power-saving for battery-based electronic devices, few have an interest in permanently-plugged Large-Scale Experimental Distributed Systems (LSEDS). However, a rapid study shows that each computer, member of a distributed system platform, consumes a substantial quantity of power, especially when those resources are idle. Today, given the number of processing resources involved in large-scale computing infrastructures, we are convinced that we can save a lot of electric power by proposing and applying "green policies". Introduced in this article, those policies propose to alternatively switch computer nodes On and Off in a clever way. Based on the analysis of some experimental large-scale system's usage, we propose a resource-reservation infrastructure which takes the energy issue into account. We validate our infrastructure on the large-scale experimental Grid'5000[1] platform and present the energy gains obtained.

1 Introdution

For a long time, energy saving for mobile distributed systems has been a matter of concern. However, for large-scale non-mobile distributed systems, which nowadays reach impressive sizes, it just begins to be taken into account.

At a time when Large-Scale Experimental Distributed Systems (LSEDS) (like Grids, Clouds...) are used more and more, driven by real-world applications

[1]Some experiments of this article were performed on the Grid'5000 platform, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other partners (http://www.grid5000.fr). This research is supported by the INRIA ARC GREEN-NET project (http://www.ens-lyon.fr/LIP/RESO/Projects/GREEN-NET/).

of increasing scale and complexity, we greatly lack studies and analyses of their usage. Some previous works on operational grids [8] show that they are not utilized at their full capacity. To better understand the stakes and potential savings induced by the scaling effects, we focus on the analyses of usage and energy of Large-Scale Experimental Distributed Systems (LSEDS).

By relying on Grid5000[1], a French experimental Grid platform, we obtain a case study from which we are able to get a comprehensive view of our concern. Grid5000 is an experimental testbed for research in distributed systems which proposes 4000 processors geographically distributed on 9 sites in France. This platform can be defined as a highly reconfigurable, controllable and monitorable LSEDS. Utilization of Grid5000 is specific and experiment-oriented. Each user can reserve some nodes in advance and then, during its reservation time, the user has exclusive access on these nodes. The users can create and deploy their own system image, collect data and reboot the nodes. The nodes are entirely dedicated to the user during his reservation.

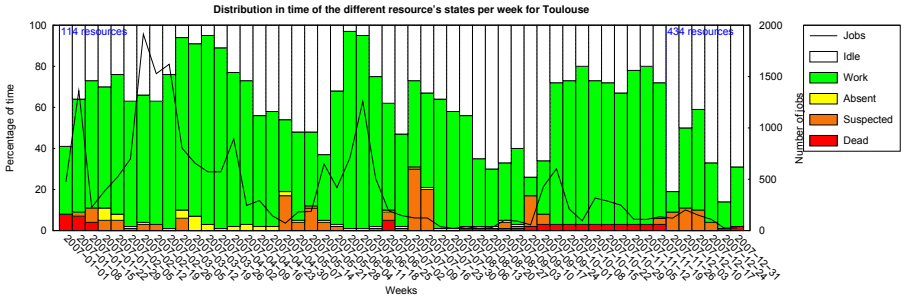

Figure 1: Global weekly diagram of usage of Toulouse Grid5000 site

We analyze the node reservation traces of Grid5000 for each site over a one-year period (the 2007 year). As an example, Figure 1 shows the usage for one site (Toulouse). The plain line indicates the number of reservations per week (we call a resource reservation "job"). For each week, we represent the time during which some cores are *dead*: (they are down), *suspected* (they do not work properly), *absent* (they do not answer), and *working* (a reservation is running). For this site, the average percentage of work time is 50.57%. A complete analysis of the usage of Grid5000 is available in [11]. We also see on Figure 1 that, during some weeks, the usage of the site can be really low. However, the real concern of such experimental platforms is to be able to support burst periods of work and communication when needed (before specific deadlines, for large-scale experiments, etc.).

Based on this analysis, we realize that the energy consumption can be reduced when the platform is not used. A framework able to control the LSEDS nodes must deal with :

- switching OFF unused nodes;

- predicting node usage to switch ON the nodes required in a near future;
- aggregating some reservations to avoid frequent ON/OFF cycles.

We propose an energy-saving model applying these aspects, and we design its implementation through the Energy-Aware Reservation Infrastructure (EARI). We design and apply a set of "green policies", introduced in this article, which propose to alternatively switch computer nodes On and Off in a clever way. Based on the analysis of some experimental large-scale system's usage, we validate our infrastructure on the large-scale experimental Grid5000 platform and present the energy gains obtained.

This paper is organized as follows. Section 2 presents our Energy Aware Reservation Infrastructure (EARI). We describe the prediction algorithms of EARI in Section 3. Our green policies, and our experimental validations are detailed in Section 4. Section 5 presents some related works, and the last Section gives our conlusions and future works.

2 Towards an Energy-Aware Large-Scale Experimental System

2.1 Architecture and context

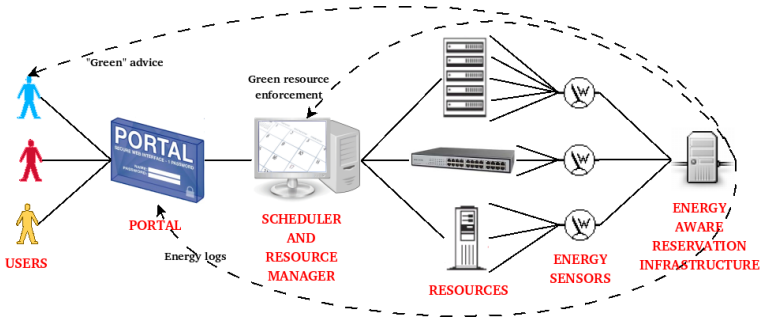

Figure 2: Global architecture

We propose a complete framework supporting energy awareness in Large-Scale Experimental Distributed Systems. Figure 2 presents the global architecture of our model. Each user is connected to a portal to submit a reservation. The scheduler processes the submission and validates it. Then, it manages the resources and gives access to them to the users who have reserved them according to the scheduler agenda. The scheduler dynamically treats the reservations when they are submitted by the user. Energy sensors monitor energy parameters from the resources (which can be nodes, routers, etc.), and these data are collected by our infrastructure. Data are used to compute "green" advices

which are sent to the user in order to influence his reservation choice. Our infrastructure also computes the consumption diagrams of the past reservations, and it sends them as a feedback on the portal in order to expose them to users. Last but not least, it decides which resources should be on and which should be off.

In our context, we assume that, for each submission of a reservation, users give their desired start time, number of resources and reservation length. When a submission occurs, the resources manager simply checks in its agenda if it can accept this reservation. If the reservation is acceptable, it is placed in the agenda when it is submitted. Actually, the resource manager does no scheduling. This submission scheme is the basic assumption of our model: the reservations are defined in terms of both time and number of resources, and the user gives a desired start time at the submission time.

2.2 Energy monitoring

We want to measure the power consumption (in Watts) of the LSEDS nodes in order to modelize the link between electrical cost and applications or processes. To measure the real consumption of some machines, we use a watt-meter furnished by the SME Omegawatt. With this infrastructure, we can make one measure per second for six different machines.

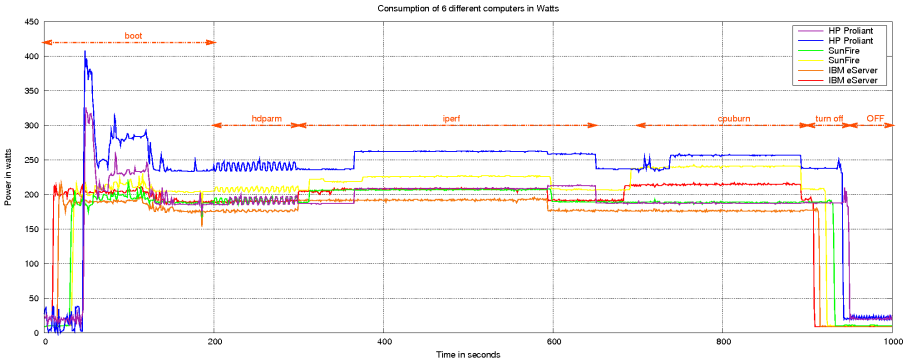

Figure 3: Electrical consumption of 6 nodes on the Lyon Grid5000 site

Figure 3 shows our results concerning the energy consumption on the Lyon site. We have dynamically collected the consumption in Watts of six different nodes representing the three hardware architectures available on the Lyon Grid5000 site: two IBM eServer 325 (2.0GHz, 2 CPUs per node), two Sun Fire v20z (2.4GHz, 2 CPUs per node) and two HP Proliant 385 G2 (2.2GHz, 2 dual core CPUs per node). We can see that the nodes have a high idle power consumption, that some of them reach impressive powers during boot, and that they consume power even when shutting down.

These experiments represent the typical life of a LSEDS node: nodes switched off but plugged in the wall socket, booting, having intensive disks accesses (`hdparm`[2]), experimenting intensive high-performance network communication (`iperf`[3]), or having intensive CPU usage (`cpuburn`[4]). We see that the `cpuburn` experiment is the most consuming and the `hdparm` one is the least consuming. We can also observe some peak of energy consumption during the `boot` and `turn off` steps (due to the intense activity of some devices (fans, hard drives...)). These results show the impact of node utilization on energy usage. Using this analysis, we design an energy-aware reservation infrastructure.

2.3 Definition of the energy efficiency model

We define a reservation R as a tuple: (l, n, t_0) where l is the length of the reservation in seconds, n is the required number of resources, and t_0 is the desired start time. We denote N the total number of resources managed by the scheduler. Therefore, we should always have $n \leq N$ and $t_0 \geq t$ where t is the actual time and $l \geq 1$ to get a valid reservation. For example, in the case of a large-scale distributed computing system, a reservation is a combination of n nodes during l seconds starting at t_0. In the case of a high-performance data transfer, a reservation is a bandwidth portion with a size of n (n can be in Mbps for example) during l seconds starting at t_0.

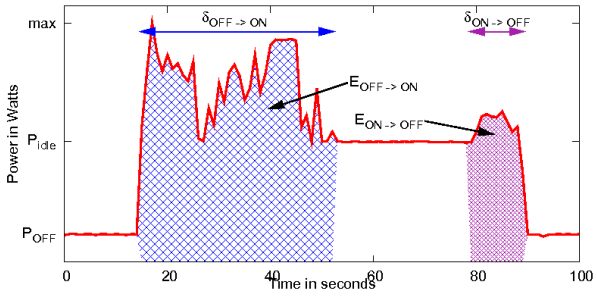

Figure 4: Booting and shutting down a resource

When a reservation is accepted by the scheduler, it is written into the corresponding agenda managed and maintained by each site. This agenda contains all the future reservations. The history contains all the past and current reservations. Therefore, when a reservation starts, it is deleted from the agenda and added to the history. P_I refers to the power consumption in Watts of a given resource (it can vary from one resource to another) when it is idle. P_{OFF} refers

[2]hdparm is a command line utility for Linux to set and view the IDE hard disks' hardware parameters.

[3]Iperf is a commonly-used network tool to measure TCP and UDP bandwidth performance that can create TCP and UDP data streams.

[4]cpuburn is a tool designed to heavily load CPU chips in order to test their reliability.

to the consumption in Watts of a given resource when it is off ($P_{OFF} < P_I$). $E_{ON \to OFF}$ ($E_{OFF \to ON}$) refers to the needed energy (in Joules) for a given resource to switch between On and Off (respectively Off and On) modes. Figure 4 illustrates these definitions. We can roughly estimate the energy consumption in Joules of a given reservation $R = (l, n, t_0)$:

$$E_m(R) = l \times \sum_{i=1}^{n} P_m(i)$$

where $P_m(i)$ is the mean consumption of the resource i.

2.4 Principle of the resource managing algorithm of our model

We split our algorithm into two parts: when a reservation is submitted (section 2.5), and when a reservation ends (section 2.6).

When a reservation arrives ($R = (l, n_0, t_0)$); at t_0, we know that there will be at least n busy resources (because of previously arrived reservations). Therefore, first of all, we check whether this reservation is acceptable, ie. $n_0 \leq N - n$. If it is not, we compute the earliest possible start time after t_0 (by taking into account the reservations which are already written down in the agenda) which is called t_1.

Then, we need to estimate different amounts of energy for the arriving reservation. We estimate the energy consumed by reservation R if it starts:

- at time t_0 or t_1 depending of the earliest available start time;
- just after the next possible end time (of a reservation) which is called t_{end};
- l seconds before the next possible start time which is called t_{start};
- at t_{slack} during a slack period (time $\geq$ 2 hours and usage under 50%).

We will see the required prediction algorithms on the next sections. In order to achieve these estimations, we need to compute t_1 (done previously), t_{end} (the end time of the next reservation after which we can put R), and t_{start} (we need to find the next reservation before which we can put R), and to estimate t_{slack} (we need to find the next slack period, see Section 3.4). Our goal is to glue the reservations in order to avoid booting and turning off nodes, which consume energy. Our infrastructure does not impose nor enforce any solution; it proposes and offers several of them to the user who can choose the most appropriate one for his reservations.

2.5 Resource allocation

In order to calculate t_{end}, we look for the next reservation end in the agenda, and we verify if it is possible to start R at that time (enough resources for the

total length). If it is not possible, we look for the next one in the agenda and so on. We then defined t_{end} as the end time of this reservation.

In the same way, we calculate t_{start} by looking for the next reservation start time in the agenda, and we check out if it is possible to place R before (this start time should then be at least at $t + l$ where t is the current time, and l is the duration of R). If the found start time does not match, we try the next one and so on. Finally, we give all these estimations (energy estimations and corresponding start times) to the user, who selects one of them. The reservation is then written down in the agenda, and the reservation number is given to the user. With this approach, the user can still make his reservation exactly when he wants to, but he can also delay it in order to save energy. It will raise user-awareness upon energy savings.

The heterogeneity of the resources available in large-scale distributed systems lead to heterogeneous energy consumptions. The scheduler allocates resources by choosing those with the smallest power coefficient. That coefficient is calculated depending on the mean power consumption of the resource (computed during reservations on a great period of time). Thus a resource which consumes a small amount of energy will have a high power coefficient and will be chosen in priority by the scheduler. This allocation policy is used when we give resources for a reservation without constraints. Actually, when the scheduler places a reservation just after another one (by using t_{end} or not) or just before another one, it allocates the resources which are already switched on (and in priority those which have the highest power coefficient). Moreover, the user can explicitly choose specific resources, so in that case, this policy is not applicable.

2.6 Resource release

When a reservation ends, M resources are freed. First of all, we compute the total actual consumption of this reservation. We give this information to the user, and we store it in the history for the prediction algorithms. Moreover, we compute the error made when we have estimated the consumption of this reservation with the corresponding start time: this is the difference between the true value and the predicted one. We will use it in the next section to compute a feedback error in order to improve our estimation algorithms.

We need to define an *imminent* reservation: it is a reservation that will start in less than T_s seconds in relation to the present time. The idea is to compute T_s such as it will be the minimum time which ensures an energy saving if we switch off the resource during this time. Actually, we define T_s so that if we switch off a resource during T_s seconds, we save at least E_s Joules (miminum fixed amount of energy). To this definition, we add a special time, denoted by T_r, which is related to the resource type and "hardware resistance". For example, mechanical hard drives can only support a limited number of ignitions. Thus we should not switch them off too often or too quickly. Therefore T_r reflects this concern and differs from one resource to another. If we denote $\delta_{tot} = \delta_{ON \to OFF} + \delta_{OFF \to ON}$, T_s is defined by:

$$T_s = \frac{E_s - P_{OFF} \times \delta_{tot} + E_{ON \to OFF} + E_{OFF \to ON}}{P_I - P_{OFF}} + T_r$$

As we can see, T_s varies from one resource to another because it depends on P_I, P_{OFF}, $\delta_{ON \to OFF}$, $\delta_{OFF \to ON}$, $E_{ON \to OFF}$, $E_{OFF \to ON}$ and T_r which depend on the resource. We can fix $E_s = 10$ Joules for example. We can notice that we should have: $T_s - \delta_{tot} \geq 0$. We want indeed to have at least enough time to switch off the resource and switch it on again. Now, we look for the freed resources that have an imminent reservation. These resources are considered as busy and are left switched on. During this active watch, we lose less than E_s Joules per resource, and then they are used again.

We look for other awake resources which are waiting for a previous estimated imminent reservation. For these m awake resources (M minus the previous busy ones and plus the other awake resources), we need to estimate when the next reservation will occur and how many resources it will take. We call this reservation $R_e = (l_e, n_e, t_e)$. We can now verify if R_e is imminent. If it is not the case, all the remaining resources are switched off. If R_e is imminent, we look for $min(m, n_e)$ resources or less that can accept this potential reservation: they are free at t_e for at least l_e seconds. We keep these resources on during $T_s + T_c$ seconds, and we switch off the other ones. T_c is a fixed value that corresponds to the mean time between the user's request and the reservation's acceptation by the scheduler (among other things, it includes the time to compute the energy estimations and a minimum time to answer to the user).

At the next reservation arrival, we will compute the estimation errors we have done, and we will use them as feedback in our prediction algorithms. Moreover, if there are idle resources (which are switched on without any reservation), and if the reservation which just arrived is not imminent, we switch off the idle resources.

3 Predictions

The efficiency of our model, compared to a simple algorithm where the resources are put into sleep state from the moment that they have nothing to do, resides in our ability to make accurate predictions of the estimation of the next reservation (length, number of resources and start time), the estimation of the energy consumed by a given reservation and the estimation of a slack period. However our prediction algorithm should remain sufficiently fast and lightweight to be applicable during reservation manager run time.

3.1 Estimation of the next reservation

First of all, we take care about the estimation of the next reservation $R_e = (l_e, n_e, t_e)$ for a given site. To estimate its start time, we calculate the average characteristics of the six previous reservations on that site.

At a given time t, we denote $R_0, \ldots, R_5$ the six previous reservations on this site (with $R_i = (l_i, n_i, t_i)$). They are the reservationse whose start times are the nearest to t (but not necessarily before t, scheduled reservations can be taken into account). These reservations are ordered by start time (R_0 is the oldest).

The estimation of the start time is done by calculating the average of the five intervals between the six previous start times. This average is added to t with the feedback to obtain the estimation:

$$t_e = t + 1/5[t_5 - t_0] + t\_feedback$$

We proceed similarly to estimate the next reservation's length and required number of resources. If we obtain $t_e < t$ (because of the feedback), we set $t_e = t + 1$. This method, based on a short past history window frame, allows a quick adaptation to the workload.

The accuracy of the next reservation's prediction is crucial for our power management. If we make too many wrong estimations, resources wait for imminent reservations that do not arrive, and so they waste energy or they are switched off just before the arrival of an imminent reservation, and so they waste the energy consumed by a halt-and-boot cycle.

3.2 Feedback on the next reservation's estimation

The feedback is used to improve the energy efficiency of our approach. As we have seen before, the estimation errors are penalizing in terms of consumed energy. We need to obtain accurate predictions.

At each reservation's arrival, we compute the estimation errors we have made. More precisely, at a given time t, the reservation $R = (l_0, n_0, t_0)$ arrives. $R_e = (l_e, n_e, t_e)$ is the last reservation that we have predicted. We denote $Err_l = (l_0 - l_e)$: the error done by estimating the length of the next reservation, $Err_n = (n_0 - n_e)$ and $Err_t = (t_0 - t_e)$ the errors done by estimating the number of resources and the start time of the next reservation respectively.

Basically, if we predict the reservation too early, then we have $Err_t > 0$. If we add Err_t to the next predicted start time, we delay the predicted start time by Err_t seconds, and that is exactly what we want to do. Then we denote $Err_l(a)$, $Err_l(b)$ and $Err_l(c)$ the three last errors for the length of a reservation. $n\_feedback$ and $t\_feedback$ are similar to $l\_feedback$:

$$l\_feedback = 1/3[Err_l(a) + Err_l(b) + Err_l(c)]$$

3.3 Estimation of a reservation's energy consumption

This estimation takes into account the user, the resource type and all the characteristics of the reservation $R = (l, n, t)$. The assumption made here is that each user has almost the same usage of the resources. What we really estimate is the average power during working time per resource for each different type of resource (different architectures for example).

3.4 Slack periods

A slack period is a period longer than two hours with a usage percentage of the platform inferior to 50%. Typically, such periods happen during the night. We chose two hours because it is just a bit longer than the average length of a reservation on Grid5000 (see [11]). Therefore a lot of reservations can take place during such a period, in terms of length. To estimate when the next slack period will occur, we use the values of the three previous days. If there was no slack period during the three previous days, we estimate that there will be no slack period that day.

To be really complete, our model should include the energy consumption of the cooling infrastructure proportionally distributed on each resource. Actually, $P_{real}(type_k, R_a)$ (the real average power for the reservation R_a for a resource which have a $type_k$ type) would include a fraction of the average power consumed by the cooling infrastructure during the duration of the reservation R_a proportionally to its heat production. However, such a fraction can be difficult to estimate, which is why most power-management systems do not take the cooling infrastructure into account. In [11], we have evaluated our prediction algorithms, and we have seen that in 70% of the cases on average, we take the good decision (to switch off the nodes or to let them on).

4 Energy gains for current and future green large-scale experimental distributed systems

To evaluate EARI, we conduct experiments based on a replay of the 2007 traces of the Grid5000 platform (these traces have been studied in [11]). Therefore, we move the reservations on a time scale by respecting several policies. Our replay mechanism works as follow: each reservation is treated when its submission time comes, so we do not need an arrival law for the reservations. We have a timer that simulates the running time of the experiment and at each second, we look into the log database to see if there is a new event, and we launch the algorithm associated to it. The events are: a reservation submission, a reservation start, a reservation end or a dead/absent/suspected resource (we put into the agenda that this resource is not available for an undetermined time).

4.1 Green Policies of EARI

We design six policies to conduct our experiments:

- *user*: we always select the solution that fits the most with the user's demand (the date asked by the user or the nearest possible date);
- *fully-green*: we always select the solution that saves the most energy (where we need to boot and to shut down the smallest number of resources);

- *25%-green*: we process 25% of the submission, taken at random, with the previous *fully-green* policy and the remaining ones with the *user* policy;
- *50%-green*: we process 50% of the submission, taken at random, with the *fully-green* policy and the others with the *user* policy;
- *75%-green*: we process 75% of the submission, taken at random, with the *fully-green* policy and the others with the *user* policy;
- *deadlined*: we use the *fully-green* policy if it doesn't delay the reservation from the initial user's demand for more than 24 hours, otherwise we use the *user* policy.

These policies simulate the behavior of real users: there is a percentage of "green" users who follow the advice given by EARI. Maybe they do not want to delay their reservation for too long, as in the *deadlined* policy. Some users do not want to move their reservation even if they can save energy by doing this; this is the *user* policy. The *fully-green* policy can illustrate the case of an administrator decision: the administrator always chooses the most energy-efficient option.

4.2 Validations on the replay of the Grid5000 traces

On the four following diagrams (Figures 5, 6, 7 and 8), we show the consumption with EARI in percentage compared with the consumption when all the nodes are always On (current case). These consumptions are computed by using the values found in Section 2. In this previous section, we have measured that $P_{idle} = 190$ Watts. Here, we will propose three different P_{idle}: 100, 145 and 190 Watts in order to anticipate some decrease in consumption in idle state mode.

The graphs present, for each P_{idle}, the results for the six different policies in order to compare their energy savings. We fix $T_s = 240$ seconds for these four diagrams (Figures 5 and 7). We also represent the ideal lowest bound which we called "all glued". It is an unreachable ideal case where we could glue all the reservations: they are all put one after the other, and the resources are switched off the rest of the time. In that case, we do not need any prediction, and thus we cannot make prediction errors. This ideal case is not reachable because it assumes that we know all the reservations in advance, yet the future is never known!

The prediction errors occur when we fail to predict an imminent reservation or when we predict an imminent reservation that does not exist. In the first case, we switch off resources that we will need in less than T_s seconds, so we lose energy. In the second case, we let some resources idle during T_s but they are useless, and they will be switched off after T_s seconds of inactivity.

Figure 5 presents the case of the Rennes Grid5000 site. As expected, in the three cases, the *fully-green* policy consumes less energy by about 5%. As expected too, the *user* policy consumes the most. We also notice that the *deadlined* policy is almost equivalent to the *50%-green* one in terms of energy consumption. We should notice that in the three cases, as P_{idle} varies, the

"100%" varies also: it corresponds to a different amount of energy in the three cases. Indeed, this "100%" is computed as follows: the time spent to work times P_{ON} plus the time spent idle times P_{idle}. The times do not change between the three cases, but P_{idle} does.

Compared to other Grid5000 sites, Rennes (Figure 5) supports a large set of long reservations. Thus the reservations are hard to move after or before other reservations. In this case, we see that our *fully-green* policy consumes the less and is really near the *all glued* bound, so our prediction models are efficient and our *fully-green* policy too. Our prediction algorithm takes the right decision (to keep the resources on or to switch them off after the end of a reservation) in 70% of the cases on average.

Aside from that, we see that sometimes our *75%-green* policy consumes more than the *50%-green* one. This is due to the random factor: we can move a small reservation that will prevent us from moving a big one at this place or that will block several others. This behavior is not energy-efficient. Therefore adding randomness does not necessarily lead to decreasing the energy consumption.

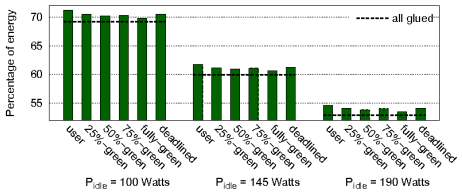

Figure 5: Energy consumption

Policy	Delayed res.	Mean delay
25%-green	44%	9 h.
50%-green	62%	9 h.
75%-green	81%	8 h.
fully-green	97%	7 h.
deadlined	73%	4 h.

Figure 6: Statistics of delayed reservations

Results of EARI for Rennes Grid5000 site with $T_s = 240$ s.

Figure 6 presents the average time that we have delayed the reservations (res. stands for reservation) and the percentage of delayed reservation in relation with the start time wished by the user. For example, the *user* policy does not delay any reservation (0% of delayed reservation and 0 hours for the mean delay time). These values does not depend on P_{idle}, but they depend on T_s, so we fix $T_s = 240$ seconds as for the previous diagram (Figure 5).

The percentage of delayed reservations for the *25%-green* policy is not 25% because of the reservations that we move. Indeed, they can take the place of the reservations we will not move (the 75% that should follow the *user* policy), so these reservations are put at the nearest date to the wished start time. We observe the same phenomenon with the *50%-green* and the *75%-green* policies. We see on Fig. 6 that in the case of Rennes, the mean delay times are really short. Compared to the cluster size (714 resources), the reservations are not really big in terms of number of resources (55 on average), so the reservations are easier to move.

The diagram (Figure 7) presents the example of Sophia. It also confirms the energy efficiency of our *fully-green* policy because it is the most efficient for

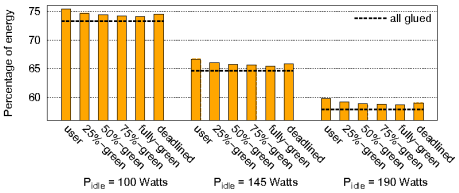

Figure 7: Energy consumption

Policy	Delayed res.	Mean Delay
25%-green	40%	8 h.
50%-green	60%	7 h.
75%-green	78%	14 h.
fully-green	94%	9 h.
deadlined	71%	5 h.

Figure 8: Statistics of delayed reservations

Results of EARI for Sophia Grid5000 site with $T_s = 240$ s.

different types of workload over a great period of time. We notice that, with our *fully-green* policy, we can save several more percentage points. These points represent a huge amount of energy at the scale of an entire cluster for a whole year.

As in the other examples, we see on Figure 8, for the case of Sophia, that increasing the number of delayed reservations does not necessarily increase the time during which they are delayed. Indeed, the 40% of delayed reservations for the *25%-green* policy are delayed by one more hour than the 60% of delayed reservations for the *50%-green* policy. We have conducted these replays on four clusters' traces studied previously (Fig. 9). These four examples represent four different workloads over a one-year period. In all the cases, our *fully-green* policy is the best one.

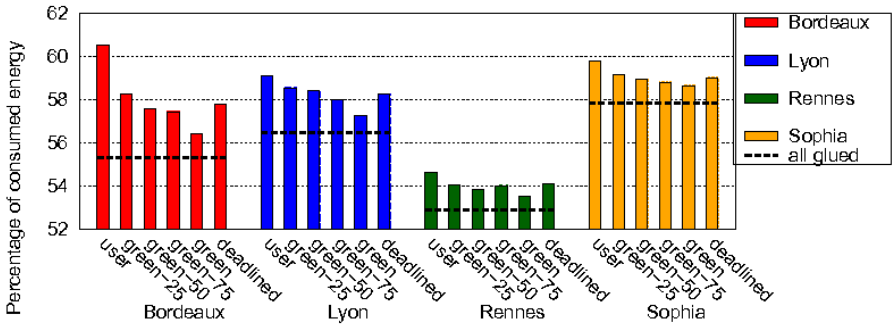

Figure 9: Energy consumption of EARI for 4 Grid5000 sites with $T_s = 240$ s. and $P_{idle} = 190$ W.

We have analysed the impact of T_s on the consumption. We can indeed change T_s to be more reactive: the resources have more idle time after the reservations to wait for the arriving imminent reservations. Thus, T_s will just be the time that the resources will wait for a reservation after the end of a reservation if our prediction algorithm has predicted an imminent reservation.

One can wonder if these switching on and off are not prejudicial to the nodes. Actually, they are already switched off and on at the beginning of each reservation since users can deploy our own image on the nodes of Grid5000. At the end of the reservations, they are rebooted on the standard image. We have seen that all the green policies are always more energy efficient than the *user* policy. All these results show that EARI can cause significant energy savings, even for the future Grids. Indeed, even 1% at the scale of a site and over a one year period represents a large amount of energy (see [12]). An extrapolation to the whole Grid5000 platform shows that for 2007, we could have saved 52% of the present consumption with our *fully-green* policy. This represents the consumption of 600 inhabitants for one year.

5 Related works

Although energy has been a matter of concern for sensor networks and battery constrained systems since their creation, energy issues are recent for full-time plugged systems. A first problem that occurs is how to measure the consumption. We have considered an external watt-meter to obtain the global consumption of a node. A different approach consists of deducing it from the usage of the node components, by using event monitoring counters [10], for example. A lot of other works on server power management based on on/off algorithms has been done [13], [3]. Some take into account thermal issues [2], [13]. The main issue in that case is to design an energy-aware scheduling algorithm with the current constraints (divisible task or not [2], synchronization [9], etc). Some algorithms include DVFS (Dynamic Voltage Frequency Scaling) techniques [9], [7] and some do not [3]. Although we are fully aware that such techniques will be available on all processors in a near future, our work does not include this in the first step presented here. Such techniques are indeed difficult to use in the presence of a processor and user heterogeneity especially if we want to design a centralized resource management algorithm. Virtualization seems to become an other promising track [6]. We have not yet spoken about network presence. A lot of works has also been done on the network level to reduce the consumption of Ethernet cards and switches' ports by adaptively modifying the link rate [4] or by turning off the ports [5]. The problem of ensuring network presence becomes more obvious with such objectives.

6 Conclusion and future work

This paper presents the first step of our work, whose goal is to better understand the usage of Large-Scale Experimental Distributed Systems and to propose methods and energy-aware tools to reduce the energy consumption in such systems. Our analysis has provided instructive results about this, using the example of Grid5000. Then, we have proposed an energy-aware model to reduce the global consumption of a large-scale experimental Grid. This infrastructure

is efficient and can be easily implemented and deployed. We have presented our results which validate our energy-aware reservation model.

We are currently working on tools, portals and frameworks proposing these results in a real-time manner to users and grid middleware. We are working on such a tool that we will integrate to the Grid5000 website. We plan to make further experiments to fully validate our infrastructure and to enhance our prediction algorithm. We also plan to make the same experiments with the whole grid traces including the grid reservation constraints (on several sites at the same time). We will study the possibility to move reservations from one site to another (according to external temperatures parameters for example). Our model is generic enough to be adapted to other infrastructure, so we are looking for logs of production/operationnal platform to validate our infrastructure in a different context.

Our long term goal is to incorporate virtualization and DVFS techniques in our infrastructure with the objective to save more energy without impacting performances. Virtualization could also solve the problem of ensuring network presence and answering basic requests from the monitoring tools of large-scale distributed systems or dealing with the high-performance data transport systems.

References

[1] F. Cappello et al. Grid'5000: A large-scale, reconfigurable, controlable and monitorable grid platform. In *6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, Seattle, Washington, USA, Nov. 2005.

[2] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *SOSP '01: 18th ACM symposium on Operating systems principles*, pages 103–116, New York, NY, USA, 2001. ACM.

[3] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA '07: Proceedings of the 34th annual international symposium on Computer architecture*, pages 13–23, New York, NY, USA, 2007. ACM.

[4] C. Gunaratne, K. Christensen, and B. Nordman. Managing energy consumption costs in desktop pcs and lan switches with proxying, split tcp connections, and scaling of link speed. *Int. J. Netw. Manag.*, 15(5):297–310, 2005.

[5] M. Gupta and S. Singh. Dynamic ethernet link shutdown for energy conservation on ethernet links. *Communications, 2007. ICC '07. IEEE International Conference on*, pages 6156–6161, 24-28 June 2007.

[6] F. Hermenier, N. Loriant, and J.-M. Menaud. Power management in grid computing with xen. In *XEN in HPC Cluster and Grid Computing Envi-*

ronments (XHPC06), number 4331 in LNCS, pages 407–416, Sorento, Italy, 2006. Springer Verlag.

[7] Y. Hotta et al. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. *IPDPS 2006*, 2006.

[8] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters. How are real grids used? the analysis of four grid traces and its implications. In *7th IEEE/ACM International Conference on Grid Computing*, Sept. 2006.

[9] R. Jejurikar and R. Gupta. Energy aware task scheduling with task synchronization for embedded real-time systems. In *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pages 1024–1037. IEEE, June 2006.

[10] A. Merkel and F. Bellosa. Balancing power consumption in multiprocessor systems. *SIGOPS Oper. Syst. Rev.*, 40(4):403–414, 2006.

[11] A.-C. Orgerie, L. Lefèvre, and J.-P. Gelas. Chasing gaps between bursts : Towards energy efficient large-scale experimental grids. In *PDCAT 2008 : The Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 381–389, Dunedin, New Zealand, Dec. 2008.

[12] A.-C. Orgerie, L. Lefèvre, and J.-P. Gelas Save watts in your grid: Green strategies for energy-aware framework in large-scale distributed systems. In *14th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 171–178, Melbourne, Australia, Dec. 2008.

[13] R. K. Sharma, C. E. Bash, C. D. Patel, R. J. Friedrich, and J. S. Chase. Balance of power: Dynamic thermal management for internet data centers. *IEEE Internet Computing*, 9(1):42–49, 2005.