Responsive Elastic Computing

Julien Perez, C. Germain Renaud, Balázs Kégl, Charles Loomis

Responsive Elastic Computing

Julien Perez
Laboratoire de Recherche en Informatique
INRIA, CNRS and Université Paris-Sud
julien.perez@lri.fr

Cécile Germain-Renaud
Laboratoire de Recherche en Informatique
INRIA, CNRS and Université Paris-Sud
cecile.germain@lri.fr

Balázs Kégl
Laboratoire de l'Accélérateur Linéaire
CNRS and Université Paris-Sud
kegl@lal.in2p3.fr

Charles Loomis
Laboratoire de l'Accélérateur Linéaire
CNRS and Université Paris-Sud
loomis@lal.in2p3.fr

ABSTRACT

Two production models are candidates for e-science computing: grids enable hardware and software sharing; clouds propose dynamic resource provisioning (elastic computing). Organized sharing is a fundamental requirement for large scientific collaborations; responsiveness, the ability to provide good response time, is a fundamental requirement for seamless integration of the large scale computing resources into everyday use. This paper focuses on a model-free resource provisioning strategy supporting both scenarios. The provisioning problem is modeled as a continuous action-state space, multi-objective reinforcement learning problem, under realistic hypotheses; the high level goals of users, administrators, and shareholders are captured through simple utility functions. We propose an implementation of this reinforcement learning framework, including an approximation of the value function through an Echo State Network, and we validate it on a real dataset.

Categories and Subject Descriptors

C.0 [**Computer Systems Organization**]: GENERAL; I.2.6 [**Learning**]: Connectionism and neural nets; I.2.8 [**Problem Solving, Control Methods, and Search**]: Scheduling

General Terms

Algorithms Measurement Performance

1. INTRODUCTION

Two approaches are currently proposed to provide computational resources at a large scale. In the grid model, institutions acquire resources and make them available to e-science users; the key point is sharing:

> What is shared, who is allowed to share, and the conditions under which sharing occurs [11].

The concept of *Virtual Organizations* (VOs) formalizes quantitative and qualitative access rights in grids. In the cloud model, the resources are leased to users, and the key point is the capacity of dynamic resource provisioning (on-demand availability), coined as elasticity by Amazon EC2. Organized sharing is a fundamental requirement for large scientific collaborations running immensely large simulations on a timescale of tens of years, such as in the High Energy Physics (HEP) community. Responsiveness, the ability to provide Quality of Service (QoS) in terms of response time, is a fundamental requirement for seamless integration of the large scale computing resources into everyday use. For instance, continuing the HEP example, a physicist wants to analyze the outputs of the above-mentioned simulations, or the future experimental events issued by the Large Hadron Collider, at its own pace. Thus, each of these infrastructures favors a specific usage scenario. As data management is the core of e-science (and business as well), exploiting the same infrastructure for acquiring, storing, and analyzing data is highly desirable.

In our research, we seek to develop resource provisioning models and operational systems that reconcile these two usage scenarios in the context of e-science. Motivated by this general goal, this paper focuses on the specific problem of supporting workloads that combine requests for quasi-immediate allocation of computational resources for a limited time period (responsive requests) and requests for computational resources whenever available (best-effort requests). The need for responsiveness arises in various situations, ranging from urgent computing applications [3], where the "limited time" might be quite high, to truly interactive applications involving computational steering, in which the individual tasks durations are extremely small [13]; another example is workflows where sequential supervision tasks are on the critical path [10].

This paper proposes an approach that uses Reinforcement Learning (RL) as a unified resource provisioning and

GMAC'09, June 15, 2009, Barcelona, Spain.

scheduling (resource allocation) mechanism. In the following, this double function (provisioning and scheduling) will be called *supervision*, and the corresponding software entity the *supervisor*. The flexibility of an RL-based system allows us to model the state of the resources, the jobs to be scheduled, and the high-level objectives of the various grid actors. RL-based supervision can seamlessly adapt its decisions to changes in the distributions of inter-arrival time, QoS requirements, and resource availability. Moreover, it requires minimal prior knowledge about the target environment including user requests and infrastructure.

We develop a general RL framework with models for classes of jobs (currently two, best-effort and responsive), for objective functions, and for the infrastructure. Furthermore we demonstrate that introducing a moderate level of elasticity in the resource provisioning is critical to ensure that both classes can coexist with a high level of user satisfaction. We considerably extend our previous work [15, 25] in the same area by first getting rid of unrealistic hypotheses about perfect knowledge of the computation characteristics, second by introducing elasticity as a key performance factor, and finally by exploiting recent advances [18] in approximating the value function through recurrent neural networks. This work has been developed in the framework of the flagship EU grid infrastructure EGEE (Enabling Grid for E-SciencE) [9], both for the grid model, and for the experimental data.

The major contributions of our paper are as follows.

- We describe a formalization of the supervision problem as a continuous action-state space, multi-objective reinforcement learning problem, under realistic hypotheses.
- We explore implementations of the reinforcement learning framework integrating those high level goals and explain the role of elastic allocation.
- We show experimentally that our RL-based supervisor achieves responsiveness without degrading utilization, as measured by several metrics related to user and administrator satisfaction.

2. PROBLEM STATEMENT

2.1 The Need for Responsiveness

Before embarking into the solution, the relevance and scope of the problem must be stated. Major industry players acknowledge interactivity as a critical requirement for enlarging the scope of high performance computing [22] and invest in this direction. Nonetheless, the general vision remains that large to massive computations dominate the e-science workloads. It might be considered as a self-realizing prediction: a long-latency software infrastructure has little appeal for tasks requiring responsiveness. The reality is more complex: because the resources and the data are there and because the workflows include long and short computations, users requiring responsiveness have little choice and do exploit the unresponsive infrastructure, albeit with repeated requirements towards improving QoS.

Considering it has tens of thousands of CPU's, petabytes of storage, an extensive coverage of scientific communities, and the perspective of sustainable development, the EGEE grid provides a good approximation of the current needs of e-science. With extensive monitoring facilities already in place, EGEE offers an unprecedented opportunity to observe and gain understanding of new computing practices of e-science.

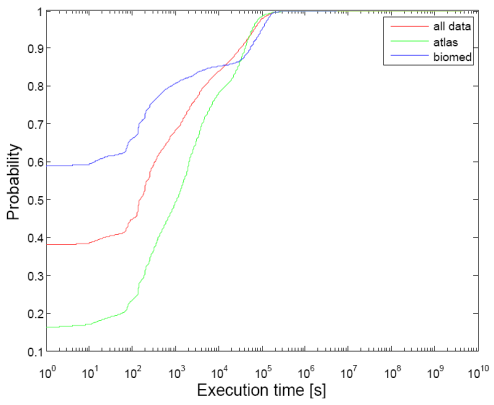

Figure 1: Cumulative distribution of execution times in the EGEE grid.

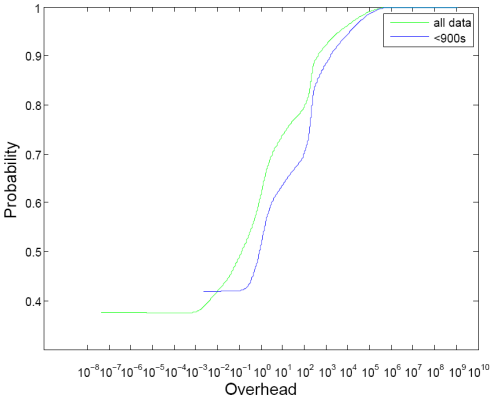

Figure 2: Distribution of the relative overhead.

The following analysis will give empirical evidence of two facts: 1) short jobs are the "dark matter" of e-science, and 2) improving responsiveness is required.

We analyze here more than one year of EGEE production under gLite [20], the EGEE middleware. The trace was provided by the Real Time Monitor project [7]. The trace covers the period November 2005 to January 2007 and includes more than 17 million production jobs belonging to 114 VOs. Jobs launched by operations management for testing service availability have been removed from the trace, thus the results faithfully describe user activity.

Fig. 2 shows the distribution of execution times from this trace. The striking feature is the importance of short jobs. All requests aggregated, the 70% quantile is approximately 900 seconds. These data also support our claim that all scientific communities need responsiveness. This is obvious for the biomedical community (Biomed VO), with more than 80% of short jobs. However, even the Atlas VO (the largest HEP community) features more than 50% of short jobs.

Fig. 2 shows the distribution of V, the dimensionless *relative overhead*; V is the ratio of the time spent in queue to the execution time of a job; the time spent in the middleware stack is not included, thus the relative overhead is

only related to the supervision issue addressed in this paper. 40% of the short jobs experience a tenfold slowdown due to queuing delays alone, a clear indicator of un-responsiveness.

2.2 Grids and clouds

The experimental data and the motivations for this paper come from the grid world. One can argue that grid infrastructures are bound to disappear into the more comprehensive cloud model. The grid-gloud convergence discussion is beyond the scope of this paper; for an in-depth comparison see [12]. However, the impact of virtualization, which is the core enabling technology of clouds, must be considered.

Virtualization brings a major advantage over grids: the "quantifiability" of allocation. A grid job must run to completion, except if checkpointing facilities are available (which is extremely rare). With the capacity to suspend, migrate, and resume computations encapsulated in virtual machines, resource allocation can be broken down along quanta of time. It might thus appear that the need for responsiveness could be addressed solely by adapting soft real-time scheduling methods [6, 28], which allocate quanta of time in order to meet the deadlines of the computations with bounded error. However, even for periodic tasks and at the modest level of concurrency offered by multi-cores the coexistence of best effort and real-time remains an open question; recent work [24] adopts a "feedback loop" method to steer the VM allocation in this case. Secondly, and more importantly, the sociology and economics of the e-science make it difficult that user computations could be embarked individually as images to draw from an undifferentiated resource pool. More likely, similarly to a recent experiment [1], the middleware stack will be virtualized and exploited onto resources leased by scientific institutions. In this last case, virtualization does not provide much help to arbitrate between responsive and best-effort requests.

3. THE RL FRAMEWORK

This section describes the RL models of the supervision problem. For the sake of completeness, the first section briefly recalls the basics of Markov Decision Process (MDP) and RL.

3.1 Markov Decision Process and RL

A Markov Decision Process is a quadruple (S, A, P, R) where S is the set of possible *states* of the system, A is the set of *actions* (or decisions) that can be taken, and P is a collection of *transition probabilities*

$$P^a_{ss'} = P\{s_{t+1} = s' | s_t = s, a_t = a\}$$

that map the current state and action to the next state. The function

$$R^a_{s,s'} : S \times A \times S \to \mathbb{R}$$

defines the *rewards* earned when moving from state s to state s' through action a.

The goal is to find a stationary policy $\pi^* : S \to A$ which chooses the action to take in each state, without knowledge of the past history (other than what is summarized in the state). The objective is to maximize the the long-term expectation of the rewards, the so-called *value function*

$$Q^\pi(s,a) = E_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a \right],$$

where $\gamma \in [0, 1]$ is a discount factor dampening the future rewards. In the scheduling context, P and R (the environment dynamics) are unknown, so the Q function has to approximated through repeated experiments. This is the definition of reinforcement learning [29]: the optimal policy will be learned by interactions with the environment.

Algorithm 1 The SARSA algorithm. $Q(s,a)$ is the value function, $\tilde{\pi}$ is the policy that selects $a^* = \arg\max_a Q(s,a)$ with probability $1-\epsilon$ and an arbitrary action a with probability ϵ, γ is the discount factor, and η is the learning rate.

```
Initialize Q(s,a) arbitrarily
s_0 ← current system state; Choose a_0 from π̃
s ← s_0; a ← a_0
REPEAT
Take action a; observe r and s'; choose a' from π̃
Q(s,a) ← Q(s,a) + η[r + γQ(s',a') − Q(s,a)]
s ← s'; a ← a'
UNTIL shutdown
```

The particular policy learning framework used in this work is based on SARSA, a member of the class of temporal-difference learning algorithm (Fig. 1). SARSA is an *on-policy* learning algorithm: the approximate value function guides the selection of the current action a, thus the reward r and the next state s'. The policy $\tilde{\pi}$ is defined by the current approximation Q. More precisely, if a^* is the action which maximizes the expected reward considering the current approximation Q (that is, $a^* = \arg\max_a Q(s,a)$), then a^* is selected with probability $1-\epsilon$. To maintain a trade-off between exploitation (using the knowledge gained so far) and exploration (looking for potentially better actions), with probability $1-\epsilon$ we select an action drawn randomly from among all the available actions. This is the so-called ϵ-greedy strategy where the parameter ϵ determines the exploration-exploitation trade-off.

3.2 The Supervision Models

We implemented two different Markov Decision Processes. The first MDP solves the problem of job scheduling under the hypothesis of a fixed amount of computing resources with the objective of minimizing the overhead and maintaining a predefined fair-share amongst VOs.

In the second MDP we consider an elastic resource and ask the MDP to also make decisions concerning the resources. The objective in this second MDP is to minimize overhead and maximize utilization. For simplicity, the fair-share constraint was dropped in this MDP, but we plan to integrate it in further work.

As explained before, a reinforcement learning formalization needs to define states, actions, and rewards for a given problem. We propose a set of variables describing states and actions to allow the formulation of the grid scheduling problem and the resource provisioning problems as continuous action-state space reinforcement learning problems. The set of variables describing the state of the system is the same in the two MDP's but they differ in the set of actions and the definition of the reward.

State Space: the grid Model

A complete model of the grid would include a detailed description of each queue and of all the resources. This would be both inadequate to the MDP framework and unrealistic:

the dimension of the state space would become very large. Instead, the state is represented by a the following set of real-valued variables:

- the total workload of running jobs;
- the time before a resource becomes available;
- the backlog, that is, the amount of work corresponding to queued jobs;
- the number of idle machines;
- the proportion of jobs of each VO in the queues.

Any job management system provides the last two descriptors at any time. The three first descriptors, associated with workload, are discussed below.

Action Space for the Scheduling MDP: the Job Model

In the first MDP each waiting job is a potential action to be chosen by the scheduler. As a consequence, except if there is no job waiting, the scheduler will always select a job when a resource become available (*greedy* allocation). A job is represented by a set of descriptors.

- the type of the job (batch/interactive);
- the VO of the user who submitted the job;
- the execution time of the job (the time to complete the job without any queuing or management overhead).

The VO associated with the job is a mandatory feature in large scale grids systems, and is available along the whole lifecycle of the job.

If the choice between batch and interactive quality of service is proposed by the grid environment, the knowledge of this type is a realistic assumption: the interactive/batch flag is known for each job before the execution, and since the user has a strong interest to correctly specify it, we can trust it.

Workload Descriptors

The first three descriptors in the state space and the last one in the action space are related to the execution time of jobs. In this work, we compare two models. The *oracle* model assumes perfect knowledge of this quantity when the job is placed in the queue. Although utterly unrealistic (except in very specific cases), this hypothesis provides an upper bound on the quality of the RL-based scheduling and resource provisioning as well. The *estimated* model makes a very crude estimate of the execution time by the median of the execution times for the batch and interactive jobs, respectively, along an extended time window. Here, the hypothesis is fully realistic: historical data are readily available online inside gLite.

Action Space for the Elastic Provisioning MDP

The decision problem associated to elastic computing extends the scheduling framework to adjusting the number of computing resources available for maximizing the utilization of the resources. Thus, there are two actions in this case: a job (to be scheduled) and a request for a specific number of processors (cores in the present setting) to be used for the next period of time. This work sticks to the grid model where jobs are not virtualized. Without virtualization, the range of adjustments is constrained: as a running job cannot be suspended (*e.g.* for going back in a queue), the number of resources must always be larger or equal to the number of running jobs.

Rewards

The reward function used in the scheduling problem is a combination of the *responsiveness utility* and the *fairness*.

The responsiveness utility is formally defined as

$$W_j = \frac{\text{execution time}_j}{\text{execution time}_j + \text{waiting time}_j}. \quad (1)$$

The responsiveness utility represents the reward associated with minimizing the overhead ($W_j = 1/(1 + V_j)$). In both the *oracle* and *estimated* models, the rewards are computed when the job completes, thus when its actual execution time and queuing delays are available. Hence, the delay separating the action and the reward is highly variable; with their short execution time, interactive jobs have a more immediate impact on the learning process.

The fairness represents the difference between the actual resource allocation and the externally defined share given to each VO. The allocation process should be such that the service received by each VO is proportional to this share. If there are n VO's, the shares are usually expressed as the n-vector of the percentages of the total resources $w = (w_1, \dots w_n)$. Let S_{kj} be the fraction of the total service received by VO k up to the election of job j. Then, the deficit distance between the optimal allocation and the actual allocation is a good measure of the unfairness. The deficit distance is defined as

$$D_j = \max_k (w_k - S_{kj})_+,$$

where $x_+ = x$ if $x > 0$ and 0 otherwise.

The unfairness is bounded above by $M = \max_k(w_k)$. A normalized fairness reward can thus be derived by a simple linear transform. If M is the maximal unfairness, the fairness utility F_j associated to the election of job j is

$$F_j = -\frac{D_j}{M} + 1. \quad (2)$$

The reward function used in the elastic computing decision problem also uses the responsiveness utility, but this time it is combined with the measure of resource *utilization*. Formally, let $(T_1, \dots, T_N)$ be the successive instants of decision making, as described in the *Action Space* section, with $T_1 = 0$. Let P_k be the number of processors allocated in the interval $[T_k, T_{k+1}]$ for $1 \leq k < N$. Finally, let f_n be the sum of the execution times of jobs completed at time T_n. The utilization reward U_n at time T_n is then defined as

$$U_n = \frac{f_n}{\sum_{k=0}^{n} P_k (T_{k+1} - T_k)} \quad (3)$$

With these definitions, all the rewards are in the $[0, 1]$ range, thus on the same scale. In both problems, the actual reward is a linear combination of two of the three rewards. In the scheduling problem, the reward is defined as

$$R_\text{s} = \lambda_\text{s} W + (1 - \lambda_\text{s}) F,$$

whereas in the elasticity problem, the reward is defined as

$$R_\text{e} = \lambda_\text{e} W + (1 - \lambda_\text{e}) U,$$

where $\lambda_s, \lambda_e \in [0, 1]$ are coefficients that allow controlling the trade-off between the rewards.

3.3 Continuous State-Action Space and Echo State Networks

In both problems, the state-action space is continuous (real valued). As a consequence, implementing the assignment $Q(s,a) \leftarrow Q(s,a) + \eta[r + \gamma Q(s',a') - Q(s,a)]$ in Fig. 1 is not immediate. The straightforward method would be to discretize the values (binning), and use a lookup table to represent $Q(s,a)$. However, the space dimensionality is high: with 7 VO's, the state-action space for the scheduling problem is $\mathbb{R}^{(19)}$.

The table representation would either require large bins (thus a very rough approximation) or a large number of bins that would result in excessively long training time. The alternative is to use a non-linear continuous approximation, as proposed in numerous works (*e.g.* [5, 31]). The design choice then lies in the interpolation method: one can use neural networks, Gaussian processes [26], or kernel methods, to cite a few of the available classes of algorithms.

Whatever method is used, the simple assignment in line 4. of the SARSA algorithm must be replaced by a learning procedure. In the case of the neural models, there are two possibilities: stochastic on-line learning, where the network is modified in each iteration only using the newly acquired training example, and batch re-learning, where the neural network is re-trained from scratch each time a new training example is added to the training set. For the time being we are using the batch option for simplicity and efficiency.

We considered two approaches for representing the value function $Q(s,a)$. The first one uses an ordinary feed-forward neural net and trains it using standard back-propagation. The second approach uses an Echo State Network (ESN) [18] from the family of recurrent neural nets. The advantage of this latter approach is that it can represent a system that goes beyond the standard Markovian assumption, in which it is assumed that all the past is entirely captured by the state descriptors. The discussion of the relevance of this choice is deferred to Section 6.

In the continuous approximation problem, learning the target Q function can be considered as an optimization problem. However an exploration-exploitation tradeoff must be ensured during the learning process to correctly sample the expectation reward function. In the case of continuous representation of the state/action space, several algorithms based on gradient descent and residual minimization has shown good efficiency and robustness in synthetic and real applications [8] [2] [16]. One of the advantages of the ESN is the simplicity of the learning algorithm: a linear regression [18] over the output weights is used to learn the target function.

In a very complex optimization landscape, running the modified SARSA algorithm with an untrained ESN would lead to extremely bad decisions in the beginning. This would adversely impact the performance both because of the actual scheduling of the first jobs and because of a poor initial approximation of the value function. To overcome this initialization issue, the RL system is pre-trained off-line with an early deadline first policy. After a few learning sweeps using the collected rewards, the network can start to take its own decisions and to be optimized using real rewards.

4. EXPERIMENTAL SETUP

We developed a simulation framework to evaluate the performance of RL-based resource allocation and scheduling. This section presents the simulation methodology, the workloads, and the experiments.

4.1 Simulation Methodology

We perform a discrete event simulation of the complete lifecycle of jobs. The events are submission, dispatch, and termination. The submission of a job adds an entry onto a shared queue. The state of the art batch schedulers heavily rely on multiple queues, with a simple FIFO scheduling inside each queue and prioritization amongst queues based on configuration files. Although our simulator can manage multiple queues, one of the goals of this work is to show that model-free methods are more effective, thus all jobs are fed to a unique queue.

The most important event is the termination of a job, which causes the manager to select a new job to run in all cases. In realistic schedulers (and in our implementation), the selection process is at worst on the scale of milliseconds, thus each termination is an opportunity to select a job. In the elastic case, a termination event might also lead to extending or contracting the resource pool. In order to create a realistic model, we added two constraints. First, the extension/contraction cannot be assumed to be instantaneous. Thus, the pool size has been constrained to be stable for at least 15 minutes. Second, the administrative structure responsible for a pool (the equivalent of site administrators in the rigid case) will be required to guarantee some basic level of service in terms of a minimum number of available cores. Thus, in our simulations, the number of cores is not allowed to drop below 30.

The core of the simulator is the learner. In the SARSA algorithm, the exploration-exploitation tradeoff parameter ϵ was set to 0.05, the discount parameter γ was set to 0.8 and the learning rate η to 0.2. The reservoir of the ESN is composed by a set of 100 sigmoidal neurons, the weights are randomly fixed in $[0, 1]$ with 10% of connectivity between the neurons of the reservoir and 15% of connectivity between the reservoir and the output neurons as in [18].

In all simulations, the first and last 500 jobs were dropped from the result, in order to avoid the bias in the results introduced by the ramp-up and draining phases.

4.2 The Input Workload

The basis for the input workload is a trace from EGEE, namely the log of the PBS scheduler of the LAL site of EGEE. The trace covers the activity of more than seven weeks (from 25 Jul 2006 to 19 Oct 2006). It includes more than 9000 user jobs, not counting the monitoring jobs which are executed concurrently with the user jobs and consume virtually no resource; they were removed from the trace. All jobs are sequential, meaning that they request only one core.

From this trace we had to decide which requests are tagged as either interactive or batch, in order to simulate a situation where such requirement for QoS would be proposed. While the submission queue could have provided some hint, most queues include jobs with the full range of execution times. This is due to the fact that the queues are mostly organized along VO's, not along quality of service. We decided to tag jobs with an execution time less than 900 seconds as interactive jobs, and the other ones as batch jobs. Otherwise,

	Size	Mean	Median	Std
Interactive	4480	160	190	108
Batch	5020	34927	15615	78755

Table 1: The EGEE workload. *Size* is the number of jobs. The statistics refer to the execution time. All times are in seconds.

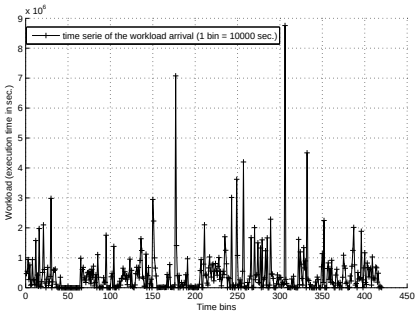

Figure 3: The service request process

the workload is kept unchanged. Table 1 summarizes the characteristics of the trace.

The trace includes the identifier of the target resource which is described in the PBS log as a core. In the period use in the experiments, the number of available cores is fairly constant ($P = 81$).

The extended timescale of the trace offers the opportunity to test the capacity of the RL-based supervisor to adapt to changing conditions. The large value of the standard deviation in Table 1 is a first indicator of high variability. Fig. 3 shows the process of *service requests*. The service request is the average of the requested CPU time over a given time interval (here 10000 seconds). Obviously, the service request is a bursty process: for instance, the peak at 300 amounts to 12 days of work for the 81 cores. However, the overall utilization remains moderate, at 0.56.

Amongst the VOs present in the trace, only six contributed significantly. The target vector is

$$[0.53, 0.02, 0.17, 0.08, 0.01, 0.16, 0.03].$$

The last share corresponds to the aggregation of the small VOs. In the segment considered in the workload, the fairness utility of the native scheduler is nearly constant (after the ramp-up phase) at 0.7.

4.3 The Experiments

We ran simulations using the workload described above with the following configurations:

- RIG-ORA - The resource configuration is rigid; the number of cores P is fixed (to 81, for comparison with EGEE). Thus, we experiment on the scheduling MDP. Moreover, the actual execution times are assumed to be known at the submission time (*oracle* model). Inside this setting, the weight λ_s of the responsiveness utility is varied: for instance, experiment RIG-ORA-0.5 corresponds to $\lambda_s = 0.5$.

- RIG-EST - The resource configuration is rigid as in the previous case, but the execution times are estimated by the median of their respective categories (*estimated* model).

- ELA-ORA and ELA-EST. The resource allocation is now elastic, and we experiment on the elastic provisioning MDP, both in the *oracle* and *estimated* models. Inside this setting, the weight of the responsiveness utility λ_e is varied from 0.5 (equal weight) to 1 (utilization not considered).

	Interactive		Batch	
	Mean	Stdev	Mean	Stdev
RIG-ORA-0.5	0.866	0.314	0.892	0.228
RIG-ORA-1	0.869	0.308	0.893	0.226
RIG-EST-0.5	0.864	0.319	0.894	0.223
RIG-EST-1	0.867	0.311	0.899	0.216
NAT	0.628	0.418	0.830	0.265

Table 2: Statistics of the responsiveness utility-Rigid.

Finally, we compare our results with the NAT experiment, which is the result of the activity of the EGEE scheduler as collected in the trace.

5. EXPERIMENTAL RESULTS

5.1 Performance Metrics

The most important performance indicators are related to 1) the performance of the RL method itself, and 2) the satisfaction of the grid actors. The quality of the optimization performed by the RL is measured by the distribution of the target indicator, which is the responsiveness utility W. Even if W can be satisfactorily optimized, it remains prove that it correctly captures the users' expectations regarding QoS. The user experience is dominated by the wallclock queuing time, which is also reported. Considering fair-share, we report the difference between the fair-share achieved by the native scheduler (as the state of the art for fair-share), and the fair-share of our scheduler; both are computed following Eq. (2). Utilization, as computed by Eq. (3), is reported directly.

5.2 The Scheduling MDP

Responsiveness Utility

Table 2 presents the summary statistics for the responsiveness utility W, and Fig. 4 shows the inverse cumulative distribution functions of W (*i.e.* $P(W > x)$ as a function of x). The first result is that the RL architecture (including the ESN) efficiently optimizes the responsiveness utility. Recall that from the definition of W in Eq. (1), the closer W is to 1, the better. Considering the summary statistics, the average responsiveness of the RL-based methods applied to interactive jobs is typically 0.86, while the native scheduler achieves only 0.63. Moreover, the standard deviation is reduced by approximately 25%. For batch jobs, the RL-scheduler does not degrade the average performance, and there is even a slight improvment. Considering the distribution (Fig. 4), W is larger than 0.9 (that is, off the optimum by 10% or less) for 82% or more of the interactive jobs. The plots have been truncated on the vertical axis for readability. The rightmost part shows that there is an empirical

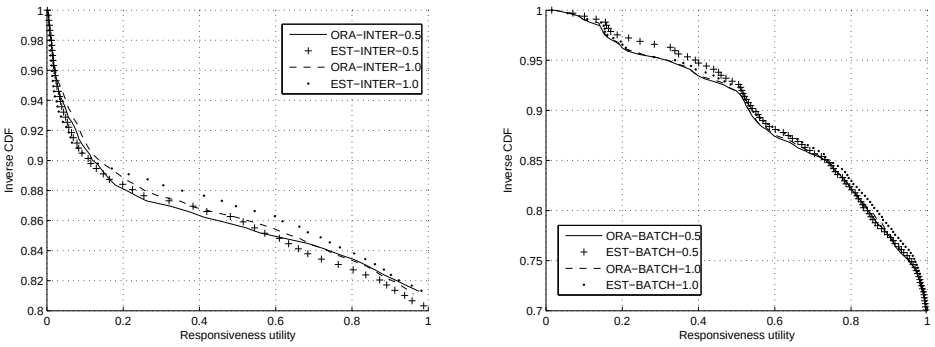

Figure 4: Distribution of the responsiveness-Rigid. Left: interactive jobs; right: batch jobs

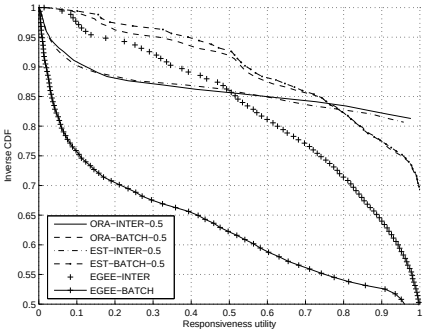

Figure 5: Comparison of RL and baseline (EGEE)-Rigid

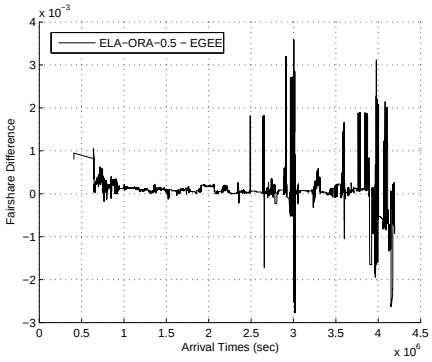

Figure 6: Dynamics of the fair-share-rigid case

	Mean	Std
RIG-ORA-0.5	1096	5137
RIG-ORA-1	862	3505
RIG-EST-0.5	1087	4838
RIG-EST-1	1152	5417
NAT	2756	8844

Table 3: Statistics of the queuing delay for interactive jobs-Rigid

threshold in the optimization process: only very few jobs can achieve a responsiveness larger than 0.98.

Considering the weight λ_s, the most surprising result is its very moderate impact. The reason is probably that, since all VOs have interactive and batch jobs, the conversion from specialized queues to one large bag of tasks creates fair-share as a side-effect.

The second result is that switching from the unrealistic oracle setting to a very simple estimation method degrades the performance only marginally. The explanation is the bimodality of the distribution of the execution times shown in Table 1. These two features, mix of interactive and batch, and bimodality, are not specific to our sample; it can be generalized as shown by the discussion of the one-year EGEE workload in Section 2.1.

Turning to the comparison with the native scheduler, Table 2 and Fig. 5 show that the RL scheduler improves massively the native scheduler for all the jobs (both interactive and batch). For the interactive case, only 53% of the jobs reach a 0.9 W in the native scheduler, versus more than 80% in the RL scheduler. A lesser but still significant improvement is reached for the batch jobs (64% vs. 77%). The batch case exemplifies once again the potential of improvement related to switching from a hard-coded priority system relying on separate queues and manual setting of complex parameters to a model-free framework. The superior performance of interactive jobs proves that the responsiveness utility was indeed a good optimization target.

Fairness

Fig. 6 shows the dynamics of the fair-share. The horizontal axis is the simulated time, and the vertical axis is the difference between the RL and EGEE fairness utilities. Only one experience has been reported, the other ones behaving very similarly. The difference is actually negligible. Most of the time, the RL scheduler is marginally superior, with some excursions corresponding to bursts.

Queuing Delay

We now consider the queuing delay (Table 3 and Fig. 7). Under the rigid scheduler, only 86% of the interactive jobs experiment a queuing delay below 2 minutes. This is a much better performance than the 63% featured by EGEE, but not in the range of true Quality of Service.

The burstiness of the service requirements, pointed in the analysis of the service request (Fig. 3), is the reason of this difficulty. At some points in time, the service request is sim-

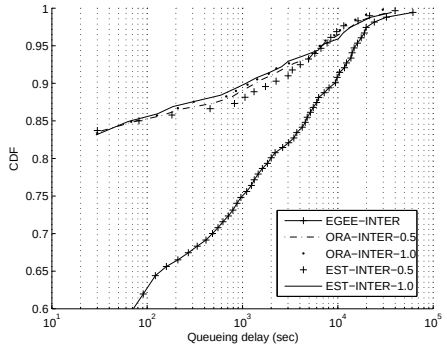

Figure 7: Distribution of the queuing delay for interactive jobs-Rigid

	Interactive		Batch	
	Mean	Stdev	Mean	Stdev
ELA-ORA-0.5	0.965	0.177	0.966	0.116
ELA-ORA-1	0.971	0.160	0.966	0.121
ELA-EST-0.5	0.958	0.184	0.960	0.123
ELA-EST-1	0.968	0.167	0.968	0.113

Table 4: Statistics of the responsiveness utility-Elastic

ply too high, while resources are unused during extended periods. In the next section, we explore a dynamic adaptation of the resource pool.

5.3 The Elastic Provisioning MDP

Responsiveness Utility

Table 4 presents the summary statistics for the responsiveness utility. Comparing with Table 2, on average, the elastic supervision allows to reach a responsiveness utility four times closer to the optimum than the rigid scheduler in the interactive case (and three times in the batch case). Moreover, the variance is also reduced.

Fig. 8 compares the inverse cumulative distribution functions of W amongst the two MDPs. The elastic provisioning MDP clearly outperforms the scheduling MDP: the elastic supervision provides a responsiveness utility for interactive

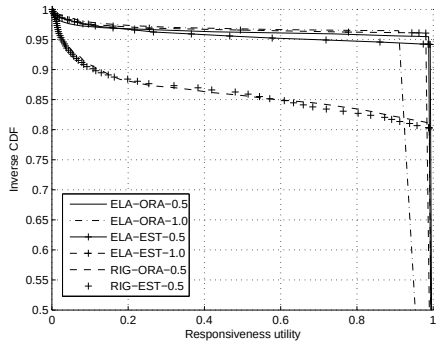

Figure 8: Distribution of the responsiveness utility for interactive jobs-Comparison of Elastic and Rigid

	Mean	Stdev	Speedup
ELA-ORA-0.5	606	4245	45%
ELA-ORA-1	236	2200	73%
ELA-EST-0.5	458	3771	58%
ELA-EST-1	414	3584	64%

Table 5: Statistics of the queuing delay for interactive jobs-Elastic

jobs above 0.9 for more than 94% of the jobs. Moreover, the behavior of the elastic MDP is nearly "flat", meaning that an overwhelming fraction of jobs achieve very comparable performance with respect to W. This rightmost part shows the same threshold as in Fig 4 (approximately 0.98 for all settings except "Estimated, $\lambda_e = 0.1$" where it is only 0.93), but with a much better probability (in the range 94%-96%), showing a very good optimization performance in absolute terms, and a very significant improvement over the rigid setting.

Elasticity and Utilization

Fig. 9 (left graph) shows the evolution of the number of cores along time. Comparing with the service requests (Fig. 3), the elastic provisioning MDP does adapt to the large burst after time 3.0E6, as well as to the low level of requests in the beginning of the trace. It is also interesting to notice that the preferred value is 90 cores, which is close, but superior, to the EGEE configuration (81 cores).

Fig. 9 shows the dynamics of the utilization (defined in Eq. 3. For clarity, only the results of the *estimated* model are displayed, the *oracle* ones being very close. The initial peak corresponds to the low activity period and the consequent reduction of the number of cores. Elastic provisioning with utilization constraint enabled (ELA-EST-0.5) reaches a nearly 100% utilization, much better than the rigid provisioning. After that, the elastic and rigid curves cross, and the price to pay for the superior responsiveness becomes apparent: the elastic utilization is asymptotically 50%, while the rigid utilization can reach a steady 70%. Significant overprovisioning seems thus to be required under moderate to high service requests for pretending to QoS. However, it must be stressed that this overprovisiong is compensated by the downsizing of the resource pool when the workload is low. This is only a qualitative asessement. We are currently exploring a more quantitative model for this

Queuing Delay

The elastic scheme significantly improves the quality of service. For interactive jobs, the distribution of the queuing delay (Fig. 10, cumulative distribution) is much more concentrated in the area below 120 seconds. More precisely, at worst 5% of the jobs are above the 2 minutes barrier. On average, the speedup is always above 45% (Table 5). Compared to the rigid case, the impact of the settings (estimation vs oracle, and scaling factor λ_e) is much more pronounced. The easiest (although unrealistic) case for elastic provisioning is $\lambda_e = 1$ (utilization not considered) and *oracle* (execution times known in advance). Dropping utilization gives a 2.5 times advantage on average over a more balanced optimization target ($\lambda_e = 0.5$). When execution durations are not known, the difference in the scaling factor has much less impact. However, estimation actually improves over perfect

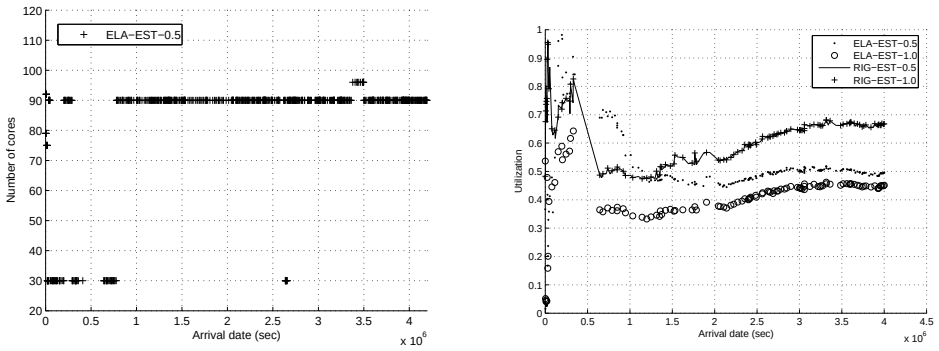

Figure 9: Dynamics of the elasticity. Left: number of cores; right: utilization

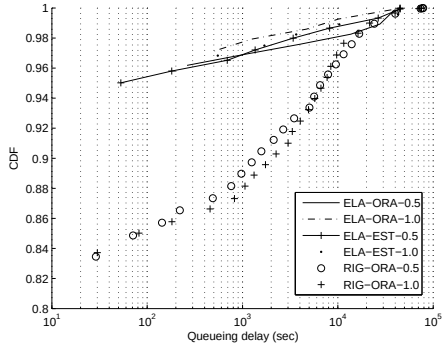

Figure 10: Distribution of the queuing delay for interactive jobs

knowledge for $\lambda_e = 0.5$.

6. RELATED WORK

The general literature related to the application of RL to scheduling is beyond the scope of this paper. The specific case of mixed workloads has been explored in depth by Tesauro, in the context of data centers [32, 30]. There are many differences between the grid and data center behavior, which explains the need for the different model presented here. The first one is the multi-objective setting. In Tesauro's work, the issue is the optimal allocation inside a fixed set of resources amongst distinct workloads, where the objective is to maximize a revenue function. In our work, the objectives (fairness and responsiveness, or utilization and responsiveness) are fully heterogeneous, and the performance has to be evaluated separately. A second difference lies in the time scales, and thus the acceptable hypothesis. Data centers serve Web requests; the arrival and service process a) can be reasonably approximated by queuing models, at least on interval of times significant with respect to the objective function, and b) feature short range correlations at worst, or are even completely memoryless (Poisson arrivals). Although not much is known yet on grid behavior, [21] and our own observations [**?**] indicate that this is not true for grid workloads. As a consequence of these two differences, our variable selection is much less parsimonious than in Tesauro's work, because we have to represent the system by aggregated observed quantities (on the numerical side) rather than by model parameters, and we have also to include categorical data (the VOs).

Tesauro has pioneered the use of neural networks as function approximators in reinforcement learning [31]. Considering scheduling and provisioning, realistic systems, whether grids or data centers, cannot be considered as Markovian, first in the intrinsic requirement processes, second because of the lags introduced by the delayed measurement in the RL learning (after job termination), and finally because of the delays introduced by re-configuring the system. These delays are crudely exemplified in our work by the 15 minutes stability. They are also described in [30] as *switching delays*. To cope with the non-Markovian behavior [30] integrates some past states in the descriptors. An alternative is to exploit recurrent (memory-enabled) neural networks. ESN were found to be particularly well suited for learning and predicting time series [19]. ESNs are also relatively easy to train, compared to other recurrent networks. Thus, embedding them inside the learning process is a promising way to address problems where the past states and actions might be relevant. Moreover, some convergence results for RL based on ESN approximators have been recently obtained [17].

7. CONCLUSION AND PERSPECTIVES

This paper shows that the combination of RL and ESN can address an issue typical of the new challenges in Machine Learning: devising an efficient policy for a large and noisy problem where no approximate model is available. The problem at hand also exemplifies a real-world situation where traditional, configuration-based solutions reach their limits, calling for autonomic methods. One of the most interesting results is the robustness of the method to crude estimations, in a situation where the variability is high.

Our future work will follow two avenues. The first one will integrate a more refined model of the switching delays, based on realistic hypothesis of future grid-over-clouds deployments. The second one will explore more aggressive methods for favoring interactive jobs when the RL-based supervision appears to be lagging behind.

Acknowledgment

This work has been partially supported by the EGEE-III project funded by the European Union INFSO-RI-222667 and by the NeuroLOG project ANR-06-TLOG-024.

8. REFERENCES

[1] Stratuslab, 2008.

[2] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.

[3] P. Beckman, S.ßNadella, N.ßTrebonß, and I.ßBeschastnikh. SPRUCE: A System for Supporting Urgent High-Performance Computing. *IFIP series*, (239):295–311, 2007.

[4] C. Blanchet, R. Mollon, D. Thain, and G. Deleage. Grid Deployment of Legacy Bioinformatics Applications with Transparent Data Access. In *7th IEEE/ACM International Conference on Grid computing*, pages 120–127, 2006.

[5] J.A. Boyan and A.W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376.

[6] A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Proc. of the 7th IEEE Real-Time Technology and Applications Symposium*, 2001.

[7] D.J. Colling and A.S McGough. The gridcc project. In *International Conference on Communication System Software and Middleware*, pages 1–4, 2006.

[8] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:219–245, 2000.

[9] F. Gagliardi et. al. Building an Infrastructure for scientific Grid computing: status and goals of the EGEE project. *Philosophical Transactions of the Royal Society A*, 1833, 2005.

[10] J. Montagnat et al. Workflow-Based Data Parallel Applications on the EGEE Production Grid Infrastructure. 6(4):369–383, 2008.

[11] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *Intl Jal Supercomputer Applications,*, 15(3):200–222, 2001.

[12] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop*, pages 1–10. IEEE, 2008.

[13] C. Germain, R. Texier, and A. Osorio. Interactive Volume Reconstruction and Measurement on the Grid. *Methods of Information in Medecine*, 44(2):227–232, 2005.

[14] C. Germain-Renaud, C. Loomis, J. Mościcki, and R. Texier. Scheduling for Responsive Grids. *Journal of Grid Computing*, 6:15–27, 2008.

[15] C. Germain Renaud, J. Perez, B. Kégl, and C. Loomis. Grid Differentiated Services: a Reinforcement Learning Approach. In *8th IEEE International Symposium on Cluster Computing and the Grid*, Lyon France, 2008.

[16] G.J. Gordon. Reinforcement learning with function approximation converges to a region. In *In Advances in Neural Information Processing Systems*, pages 1040–1046. The MIT Press, 2001.

[17] I. Szitaßand V. Gyenes and A. Lorinczß. Reinforcement Learning with Echo State Networks. In *Artificial Neural Networks, ICANN 2006*, pages 830–839. Springer, 2006.

[18] H. Jaeger. Adaptive nonlinear system identification with Echo State Networks. In *Advances in Neural Information Processing Systems 15*, pages 593–600. MIT Press, 2003.

[19] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78 – 80, 2004.

[20] E. Laure and al. Programming the Grid with gLite, 2006.

[21] H. Li and M. Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Perform. Eval. Rev.*, 34(4):59–70, 2007.

[22] I. Mirman. Going parallel the new way, 2006.

[23] J. Mościcki, M.T. Bubak, H.C. Lee, A. Muraru, and P.M.A. Sloot. Quality of service on the grid with user level scheduling. In *Cracow Grid Workshop*, pages 119–129, 2007.

[24] S. M. Park and M. Humpfrey. Feedback-controlled resource sharing for predictable e-science. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE, 2008.

[25] J. Perez, C. Germain Renaud, B. Kégl, and C. Loomis. Utility-based Reinforcement Learning for Reactive Grids. In *The 5th IEEE International Conference on Autonomic Computing*, 2008. Short paper.

[26] C. E. Rasmusen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[27] Q. Snell, M.J. Clement, D.B. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-scheduling. In *IPDPS '00/JSSPP '00: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 137–153. Springer-Verlag, 2000.

[28] A. Srinivasan and J. H. Anderson. Efficient scheduling of soft real-time applications on multiprocessors. *Jal. Embedded Computing*, 1(3):1–14, 04.

[29] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[30] G. Tesauro, N. K. Jong, R. Das, and M. N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[31] G. Tesauro and T.J Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390, 1989.

[32] Gerald J. Tesauro and Jeffrey O. Kephart. Utility functions in autonomic systems. In *Proceedings of the 1st International Conference on Autonomic Computing(ICAC'04)*, pages 70–77, 2004.