Multi-objective reinforcement learning for responsive grids

Julien Perez, Cecile Germain-Renaud, Balázs Kégl, Charles Loomis

Noname manuscript No.
(will be inserted by the editor)

Multi-objective Reinforcement Learning for Responsive Grids

Julien Perez · Cécile Germain-Renaud · Balazs Kégl · Charles Loomis

Received: date / Accepted: date

Abstract Grids organize resource sharing, a fundamental requirement of large scientific collaborations. Seamless integration of grids into everyday use requires responsiveness, which can be provided by elastic Clouds, in the Infrastructure as a Service (IaaS) paradigm. This paper proposes a model-free resource provisioning strategy supporting both requirements. Provisioning is modeled as a continuous action-state space, multi-objective reinforcement learning (RL) problem, under realistic hypotheses; simple utility functions capture the high level goals of users, administrators, and shareholders. The model-free approach falls under the general program of autonomic computing, where the incremental learning of the value function associated with the RL model provides the so-called feedback loop. The RL model includes an approximation of the value function through an Echo State Network. Experimental validation on a real dataset from the EGEE grid shows that introducing a moderate level of elasticity is critical to ensure a high level of user satisfaction.

Keywords Grid Scheduling · Performance of Systems · Machine Learning · Reinforcement Learning

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

Julien Perez
Univ. Paris-Sud and CNRS
E-mail: julien.perez@lri.fr

Cécile Germain-Renaud (corresponding author)
Tel +33 1 69 15 42 25
E-mail: cecile.germain@lri.fr

Balazs Kégl
Univ. Paris-Sud and CNRS
E-mail: kegl@lal.in2p3.fr

Charles Loomis
Laboratoire de l'Accélérateur Linéaire (LAL) and CNRS
E-mail: loomis@lal.in2p3.fr

1 Introduction

Two approaches are currently proposed to provide computational resources at a large scale: the grid and the cloud. In the grid model, institutions acquire resources and make them available to e-science users; the key point is sharing, as stated by Foster et al. in [15]: *"resource providers and consumers defining clearly and carefully just what is shared, who is allowed to share, and the conditions under which sharing occurs."* 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, defined as the ability to allocate resources on-demand, 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 LHC, at his 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.

As a consequence, the Grid-Cloud convergence is actively experimented, mainly in the IaaS (Infrastructure as a Service) paradigm. Quoting the StratusLab initiative [27] *"Using cloud technologies for resource provisioning would enhance failover and redundancy solutions, provide elastic sites able to expand available resources, and permit machine migration for flexible load balancing"*. Discussing the Grid-Cloud convergence is beyond the scope of this paper; for an in-depth comparison see [16,19]. A key point is that 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. A more likely path is that the middleware stack will be virtualized and deployed onto resources leased by scientific institutions.

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), on *moderately elastic sites*.

The exploitation model of e-science infrastructures (High Performance Computing centers, and Grids) is dominated by the best-effort scenario, with high job throughput as the primary performance metric. The sites' batch schedulers are responsible for resource allocation, with a preliminary step of matchmaking in large grids. Batch schedulers efficiently serve complex fair-share patterns, at the expense of complex manual configuration. Although, in principle, they support advance reservations, this mechanism is rarely enabled, both because of the associated utilization problems [33] caused by the need to block best effort jobs in order to reserve slots for the future reservation, and also because the advance reservation simply does not match the request for unplanned access. The batch schedulers also provide basic facilities for immediate execution, *e.g.* in the PBS scheduler. In a previous work [17], we showed that these capabilities can be combined as an enabling mechanism for Virtual Reservations (VRes), which provides responsiveness without jeopardizing utilization. Practical experience in a supportive environment proved that the complexities of the implied configuration

severely limits the applicability of this scheme. Autonomic computing comes into play here in that 1) high-level goals such as responsiveness and fair-share should be easily tunable by system administrators, and 2) the underlying scheduling mechanism should be able to self-adapt to the uncertainties in the environment and the trends in usage.

This paper proposes an approach that uses Reinforcement Learning (RL) as a unified resource provisioning and 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 scheduling can seamlessly adapt its decisions to changes in the distributions of inter-arrival time, quality of service requirements, and resource availability. Moreover, it requires minimal prior knowledge about the target environment including user requests and infrastructure.

We develop a general Reinforcement Learning framework with models for classes of jobs (currently two, best-effort availability and responsiveness), for objective functions, and for the infrastructure. Furthermore we state 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 [18,30] 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 [21] 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) [13] 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

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 task durations are extremely small [17]; another example is workflows where sequential supervision tasks are on the critical path [14]. Major industry players acknowledge interactivity as a critical requirement for enlarging the scope of high performance computing 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

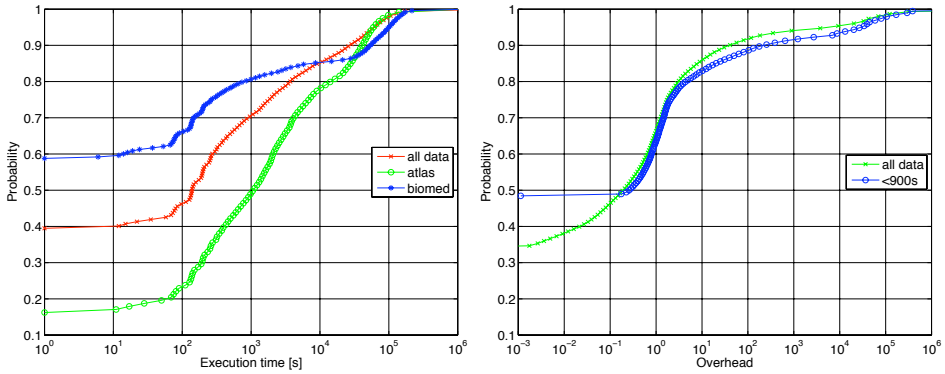

Fig. 1 Distribution of execution time (left graph) and overhead (right graph) in the EGEE grid.

responsiveness have little choice and do exploit the unresponsive infrastructure, albeit with repeated requirements towards improving quality of service.

With extensive monitoring facilities already in place, the EGEE grid offers an unprecedented opportunity to observe and gain understanding of new computing practices of e-science. Considering it has tens of thousands of CPU's, petabytes of storage, an extensive coverage of scientific communities, and the perspective of sustainable development, EGEE provides a good approximation of the current needs of e-science. 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 [12], the EGEE middleware. The data are provided by the Real Time Monitor tool [7]. They cover all the activity of the EGEE grid monitored by gLite in the period November 2005 to January 2007 and include more than 17 million production jobs belonging to 114 Virtual Organizations (VOs). Jobs launched by operations management for testing service availability have been removed, thus the results faithfully describe user activity. Fig. 1 (left) shows the distribution of execution times. The striking feature is the importance of short jobs. All requests aggregated, the 70% percentile is approximately 900 seconds. These data also support our claim in the introduction 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. 1 (right) shows the cumulative probability 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. The fact that 40% of the short jobs experience a tenfold slowdown due to queuing delays alone indicates clearly the unresponsiveness of the system. Thus minimizing the relative overhead will be target of the supervisors described in the paper.

2.2 Configuration-based Responsiveness

The VRes experience is a case for autonomic computing. Responsiveness is a major requirement of many of the mainstream users of the EGEE grid (HEP, Life Sciences)

[5]. VRes [17] has been developed and validated in EGEE. An implementation called the SDJ (Short Deadline Jobs) has been developed for gLite. A SDJ job requests immediate execution by raising the SDJ flag in its submission file. The Workload Management System (WMS) of gLite directs SDJ jobs to sites that have configured their batch scheduler for Virtual Reservation. The modification of the WMS was minor, and is now part of the gLite standard distribution. The reader is referred to [17] for more details about the configuration, and how abuse of the system can be prevented. The SDJ feature is regularly used in demonstrations (one cannot make the audience wait while the jobs are queued) and by smart users [4, 9, 11]. Nonetheless, only a handful of sites have actually enabled Virtual Reservation. The explanation lies in the "expected return of investment" for site administrators. Enabling Virtual Reservation requires modifying the configuration files of the local batch scheduler, which have been carefully crafted and stabilized to answer users and institutions requirements related to access rights and shares.

An interesting byproduct of this development is the consensus of the community on what level of responsiveness could reasonably be expected. Some middleware penalty is the unavoidable counterpart of a large scale system; a typical delay of two minutes is thus considered acceptable.

Beyond the VRes example, the lack of responsiveness, and more generally the need for flexible prioritization, together with the independent problem of fault management presently lead to an increasing usage of overlay schedulers exploiting placeholder jobs in EGEE and in TeraGrid as well. (For an in-depth presentation of this strategy, see [29].)

3 The RL Framework

This section describes the Reinforcement Learning model 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 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 [34]: 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
$s_0 \leftarrow$ current system state; Choose a_0 from $\tilde{\pi}$
$s \leftarrow s_0$; $a \leftarrow a_0$
REPEAT
Take action a; observe r and s'; choose a' from $\tilde{\pi}$
$Q(s, a) \leftarrow Q(s, a) + \eta[r + \gamma Q(s', a') - Q(s, a)]$
$s \leftarrow s'$; $a \leftarrow a'$
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 (Algorithm 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 *Scheduling MDP* solves the problem of job scheduling under the hypothesis of a fixed amount of computing resources with the objective of minimizing the overhead (as defined in Section 2.1) and maintaining a predefined fair-share amongst groups of users. In the *Elastic Provisioning MDP* we consider that the amount of computing resources may vary within reasonable bounds; we then ask the MDP to also make decisions about resource provisioning. 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 five 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.

Is this model realistic? Any job management system provides the last two descriptors at any time. The first three descriptors are related to the execution time of jobs, thus will be discussed after the definition of the job model.

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. Thus, the action space is the set of waiting jobs. Defining the action space as the set of waiting jobs implies that the scheduler will always select a job (if any) when a resource becomes available. Thus, the scheduler is work-conserving. On the other hand, if dependencies exist amongst jobs (*e.g.* if they are part of a workflow, or if co-allocation is required), they should be resolved before the jobs are enqueued, as all queued jobs are assumed to be eligible for running.

More precisely, a job is represented by two discrete and one real-valued variables:

- 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).

According to the SARSA algorithm (Algorithm 1), the selected job is, with high probability, the one that maximizes the value function, which is the expected long-term discounted reward (see below for the definition of the rewards and the computation of the expectation), or with low probability, a random one (ϵ-greedy strategy).

Is this model realistic? The VO associated with the job is a mandatory feature in large scale grids systems, and is available along the whole life-cycle of the job. The interactive/batch attribute describes an input tag requesting higher quality of service. Qualitatively, a user should tag a job as interactive when it is urgent, meaning that it should not wait in queue . Of course, the counterpart is that such jobs should not last long, and will be killed otherwise. If the choice between batch and interactive quality of service is proposed by the grid environment, the knowledge of this attribute is a realistic assumption: the interactive/batch tag will be known for each job before the execution, and since the user has a strong interest to correctly specify it, we can trust it. The interactive attribute has no immediate relation with the gLite INTERACTIVE job type, but is inspired from to the SDJ attribute.

The binary attribute (batch/interactive) could be extended to more classes. However, in the RL framework, a meaningful finer class segmentation should define a differentiated set of rewards (utility functions), for which we have no convincing examples.

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.

In order to create a realistic model, we added two constraints. First, the extension/contraction cannot be assumed to be instantaneous. Thus, the decisions concerning the size of the resource pool (number of idle machines in the state space) are taken at a fixed frequency. Second, the administrative structure responsible for a pool (equivalent of the 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 the number of cores will have a non-null lower bound. Both parameters, frequency and lower bounds, are constants of the model.

More on the Workload Descriptors

In this work, we compare two sub-models. The *oracle model* assumes perfect knowledge of the execution time of a job when placed in the queue. The *estimated model* estimates the execution time by its median, separately for the batch and interactive jobs, along an extended time window in the past. Although utterly unrealistic (except in very specific cases), the *oracle model* provides an upper bound on the quality of the RL-based scheduling, and resource provisioning as well. The *estimated model* is fully realistic: historical data are readily available online from job management systems. The motivation for choosing a very crude estimator is discussed in Section 7. One of the goals of this work is precisely to show that it is efficient.

Rewards

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

The responsiveness utility for job j is formally defined as

$$W_j = \frac{\text{execution time}_j}{\text{execution time}_j + \text{waiting time}_j}. \tag{1}$$

The responsiveness utility represents the reward associated with minimizing the overhead V_j presented in Section 2.1, as $W_j = (1 + V_j)^{-1}$. 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 some predefined 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

$$F_j = 1 - \frac{D_j}{M}. \tag{2}$$

Some VOs may ask for less than their share. Without greedy allocation, the previous rule leads to resource underutilization, a highly undesirable property. This classical problem has been addressed in the framework of network allocation as well as for processor allocation [24]), with the objective of fair excess allocation: if excess resources do exist, they should be proportionally allocated to the active requests. These methods could be adapted to our framework, by dynamically adjusting the w_k as a function of the actual requests. However, with greedy allocation, there is no risk of resource underutilization (as far as there is enough overall work). On the other hand, the excess resource can be advantageously exploited by favoring the user utility in the short term. Thus we keep the fairness utility as defined in Eq. 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 $1 \leq n < N$. 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

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

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

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

whereas in the elasticity problem, the reward is defined as

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

where $\lambda_\mathrm{s}, \lambda_\mathrm{e} \in [0, 1]$ 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 Algorithm 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.* [6,37]). The design choice then lies in the interpolation method: one can use neural networks, Gaussian processes [31], 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) [21] from the family of recurrent neural nets. The advantage of this latter approach is that this network has a memory so 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 7.

In the continuous approximation problem, learning the target Q function can be considered as an optimization problem. However an exploration-exploitation trade-off 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 have shown good efficiency and robustness in synthetic and real applications [8,**?**,**?**]. One of the advantages of the ESN is the simplicity of the learning algorithm: a linear regression [21] 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.

Table 1 Synthetic workload configurations.

Interactive	μ ($\times 10^{-4}$)	λ ($\times 10^{-2}$)	Mean exec. time (minutes)	Simulated duration (hours)
20%	2.48	1.23	67	136
40%	5.68	2.81	29	59
50%	7.70	3.81	22	44

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 life-cycle of jobs. The simulator is written in MATLAB, which greatly helped rapid prototyping of the learning components. The events are submission, dispatch, and termination. The submission of a job adds an entry onto a shared queue: although our simulator can manage multiple queues, one of the goals of this work is to show that model-free methods are more effective; the state of the art batch schedulers rely heavily on multiple queues and prioritization amongst queues based on configuration files.

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. The pool size is constrained not to drop below 30 cores, and to be stable for at least 15 minutes.

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 [21].

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: the small number of jobs in these phases is not representative of the continuous process of arrival and departures.

4.2 The Input Workload

We analyze two workloads. The first one is the traditional M/M/P queue, and the second one is extracted from real EGEE traces.

	Interactive	Batch
No. of jobs	4480	5020
Mean (s)	160	34927
Median (s)	190	15615
Std (s)	108	78755

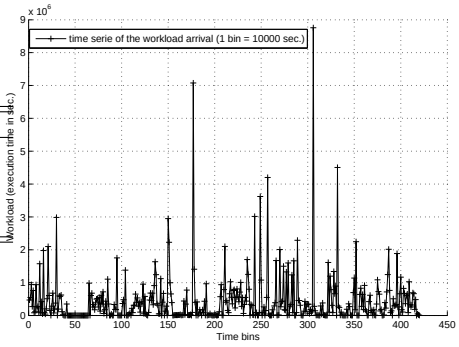

Fig. 2 The EGEE workload. In the table, the statistics refer to the execution time (in seconds). The figure shows the service request process.

The synthetic workload

The arrival process is thus Poisson with parameter λ and the execution times are exponentially distributed with parameter μ. The so-called *utilization factor* $\rho = \lambda/\mu$ must be less 1 in order to get a finite queuing time. The utilization factor controls the system load. In the following, ρ is set to 0.99. The system is thus heavily loaded, which allows the RL algorithm to demonstrate its superior performance.

The definition of interactive jobs is set to jobs with an execution time less than 15 minutes. The proportion of interactive jobs is varied across the experiments. The value of μ follows immediately from the exponential distribution $P(X > t) = e^{-\mu t}$. For a given ρ, λ is then computed as $\mu\rho P$, where P is the number of processors. In this experiment, P is set to 50. For all the experiments, 6000 jobs are simulated. Table 1 gives the resulting configurations; the first column gives the fraction of interactive jobs in the workload.

The EGEE workload

The basis for the input workload is a log from EGEE, namely the log of the PBS scheduler of the LAL site of EGEE. The number of jobs, as well as the variety of VO present in the log, support the empirical knowledge that this site is representative of a profile of general usage of EGEE. Moreover, the PBS log records all EGEE jobs, whether created through the Pilot Job scheme or by gLite. The trace has been selected from more than a year's worth of logs, in order to discover a segment where a) the load and the mix of interactive and best-effort jobs is significant with respect to the optimization goals, b) the machine pool is stable and c) the distribution of the jobs with respect to the VOs is representative of the target fair-share. The trace covers the activity of more than seven weeks (25 July 2006–12 September 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 resources; they were removed from the trace. All jobs are sequential, meaning that they request only one core. Fig. 2 summarizes the characteristics of the trace.

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, the workload is kept unchanged. 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 fig. 2 is a first indicator of high variability. The graph in Fig. 2 shows the process of *service requests*. The service request is the average of the requests for CPU time over a given time interval (here 10000 seconds). Obviously, the service request is a bursty process: for instance, the peak at bin 300 amounts to 12 days of work for the 81 cores. However, the overall utilization remains moderate, at 0.5623.

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 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 to be proved that it correctly captures the users' expectations regarding Quality of Service. The user experience is dominated by the wall-clock queuing time which is also reported. Considering fair-share, we report the difference between the fair-share achieved by the baseline scheduler (FIFO for the synthetic workload, and native for the EGEE workload, which is the state of the art in the domain) and the fair-share of our scheduler computed following Eq. 2. Utilization, when relevant, is reported directly as computed by Eq. 3.

5 Experimental Results: The Synthetic Workload

This experiment considers only the rigid case (scheduling MDP) and the oracle setting. The goal is to show that RL is a good candidate for providing responsiveness and to focus on the fair-share performance. We compare the performance of our method with a baseline one, FIFO scheduling. The same input files, created from the parameters described in Table 1, have been used for both methods.

5.1 Feasible Schedule

In this experiment, the fair share configuration is 4 VOs, with respective target weights 0.7, 0.2, 0.05 and 0.05. The schedule is feasible, meaning that the actual proportions of

Table 2 Waiting time (in seconds) for the synthetic workload with a feasible schedule.

	interactive						batch					
Inter	FIFO			RL			FIFO			RL		
active	mean	std	max	mean	std	max	mean	std	max	mean	std	max
20%	923	552	2361	108	123	975	825	539	2383	103	112	1040
40%	690	321	1425	50	58	597	642	314	1426	454	49	515
50%	740	368	1577	38	42	368	718	360	1550	343	38	397

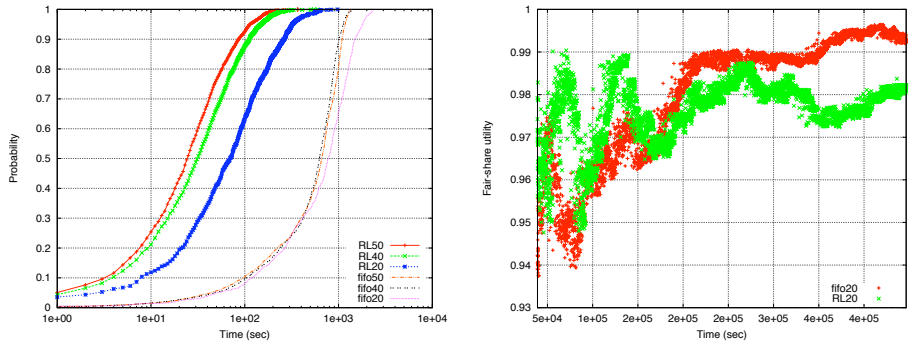

Fig. 3 Performance comparison for the feasible schedule under RL and under FIFO: (left) cumulative distribution of the queuing delay and (right) dynamics of the fair share.

work in the overall synthetic workload are the same as the target ones. Besides, inside each class of jobs (interactive and batch), the proportions are also close to the target.

The statistics of the waiting time are summarized in Table 2. The first column gives the fraction of interactive jobs in the workload, as in Table 1.

These results are quite good. The RL-method clearly outperforms FIFO: the delay is divided by more than 8 when there are 20% of interactive jobs, and by nearly 20 for the 50% case. In fact, this improvement holds, with rather similar values, for both the interactive class and the batch class. One can suspect that favoring interactive jobs results in nearly starving some batch ones. In fact, the standard deviation and the maximum are also reduced by the RL method, which proves that this is not the case.

The cumulative distribution function of the waiting time is shown on Fig. 3 (left) for the interactive class. An important result is that the delay is now acceptable for human interaction: in the worst case (20% of interactive jobs), 90% do not wait more than 2 minutes.

Fig. 3 (right) shows an example of the dynamics of the fair-share performance where the horizontal axis is the simulated time and the vertical axis, the fair-share utility. For readability, only the first experiment (20% of interactive jobs) is reported . With a feasible schedule, in the long run, the job sample is conform to the target, thus the FIFO scheduler achieves the requested fair share. Not surprisingly, the RL-method is inferior to FIFO in the long run. However, the price to pay is extremely small: in both cases, the RL method is only 3% off the ideal allocation. Besides, the RL method converges reasonably fast, considering the grid time scale: at time 5×10^4 s (13 hours), the fair share utility is above 94%. The figures for the other cases (40% and 50% of interactive jobs) are quite similar, thus omitted.

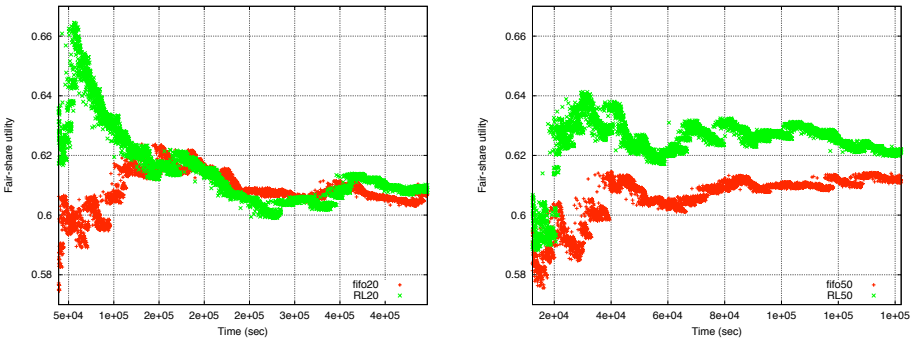

Fig. 4 Dynamics of the fair share with an unfeasible schedule: (left) 20% of interactive jobs and (right) 50% of interactive jobs.

5.2 Unfeasible Schedule

The high level objectives defined by humans may be unrealistic. It is well known that this is often the case for fair share. The target weights describe the activity of users as expected by administrators, which may significantly differ from the actual one. In this experiment, we consider the case where the target weights are 0.4, 0.2, 0.2 and 0.2, while the actual weights remain 0.7, 0.2, 0.05 and 0.05. This is an unfeasible schedule, because the first VO does not provide enough load, and the third and fourth ones ask for more resources than they are entitled to. Nonetheless, the overall load remains compatible with the resources: the pre-set utilization factor is the same, 0.99. In fact, the data-set is the same as in the previous experiment; only the weight parameters in the fair share utility function are modified.

The issue here is to assess the robustness of the RL-method in presence of unfeasible constraints. According to Eq. 2, the maximal positive distance is $0.2 - 0.05 = 0.15$, and the upper bound for unfairness is 0.4, thus the best possible schedule gives a reward of 0.625. Fig. 4 shows that the RL and FIFO achieve comparable and nearly optimal performance in this challenging case. The results about user related metrics are very similar to the feasible case, thus we do not repeat them.

6 Experimental Results: The EGEE Workload

6.1 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). We experiment on the scheduling MDP. 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 from 0 to 1. For instance, experiment RIG-ORA-0.5 corresponds to $\lambda_s = 0.5$.

Table 3 Statistics of rigid supervisor, EGEE workload.

	Interactive Responsiveness		Batch Responsiveness		Queuing delay (seconds)	
	mean	std	mean	std	mean	std
RIG-ORA-0.5	0.866	0.314	0.892	0.228	1096	5137
RIG-ORA-1	0.869	0.308	0.893	0.226	862	3505
RIG-EST-0.5	0.864	0.319	0.894	0.223	1087	4838
RIG-EST-1	0.867	0.311	0.899	0.216	1152	5417
NATIVE	0.628	0.418	0.830	0.265	2756	8844

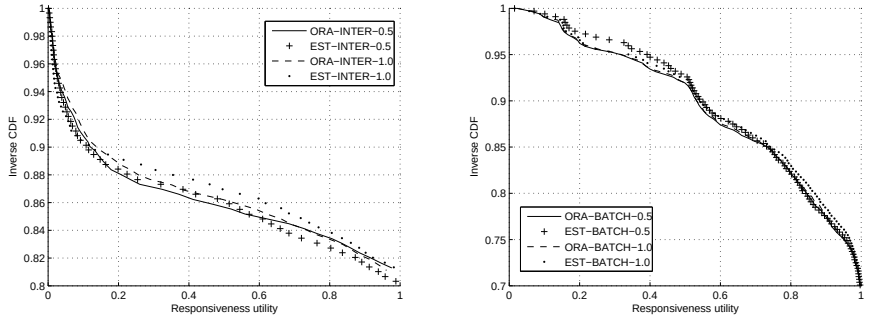

Fig. 5 Distribution of the responsiveness for the rigid supervisor: (left) interactive jobs and (right) batch jobs.

- 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).

Finally, we compare our results with the result of the native PBS scheduler, as recorded in the trace, labelled NAT in the Figures and Tables.

6.2 The Scheduling MDP

Table 3 presents the summary statistics for the responsiveness utility W, and Fig. 5 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 (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 improvement. Considering the distribution (Fig. 5), W is larger than 0.9 (that is, off the optimum by 10% or less) for 82% or more

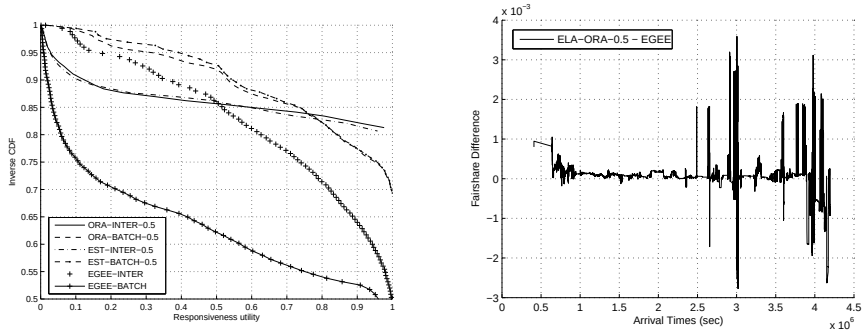

Fig. 6 Comparison of RL-Rigid and native schedulers. Left graph: inverse CDF of the responsiveness. Right graph: dynamics of the fair-share.

of the interactive jobs. The plots have been truncated on the vertical axis for readability. The rightmost part shows that there is an empirical 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 Fig. 2. 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 3 and Fig. 6 (left) 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.

We now consider the queuing delay (Table 3 and Fig. 7). As discussed in Section 2.2, the 2 minute delay is a good landmark for assessing the potential satisfaction (or lack thereof) of the interactive users. Under the rigid scheduler, only 86% of the interactive jobs experience a queuing delay below 2 minutes, a much better performance than the 63% featured by EGEE, but not fully satisfactory. The bustiness of the service requirements, pointed out in the analysis of the service request (Fig. 2), explains the difficulty. At some points in time, the service request is simply too high, while resources are unused during extended periods. In the next section, we explore a dynamic adaptation of the resource pool.

We finally examine the fairness performance. Fig. 6 (right) 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 experiment has been reported, the other behaving very similarly. The difference is actually negligible. Most of

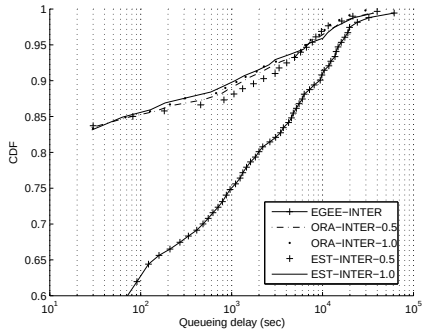

Fig. 7 Distribution of the queuing delay for interactive jobs-Rigid.

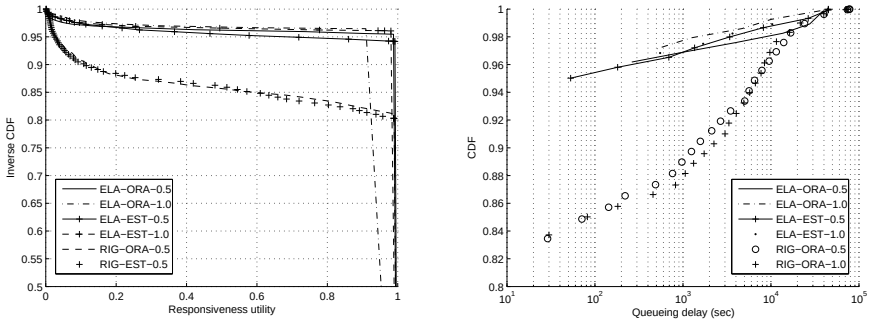

Fig. 8 Performance of the Elastic supervisor: (left) distribution of the responsiveness utility for interactive jobs-Comparison of Elastic and Rigid and (right) distribution of the queuing delay.

the time, the RL scheduler is marginally superior, with some excursions corresponding to bursts.

6.3 The Elastic Provisioning MDP

Table 4 Statistics of the elastic supervisor, EGEE workload.

	Interactive responsiveness		Batch responsiveness		Queuing delay (seconds)	
	mean	std	mean	std	mean	std
ELA-ORA-0.5	0.965	0.177	0.966	0.116	606	4245
ELA-ORA-1	0.971	0.160	0.966	0.121	236	2200
ELA-EST-0.5	0.958	0.184	0.960	0.123	458	3771
ELA-EST-1	0.968	0.167	0.968	0.113	3584	64

Table 4 presents the summary statistics for the responsiveness utility. Comparing with Table 3, on average, the elastic supervision allows to reach a responsiveness util-

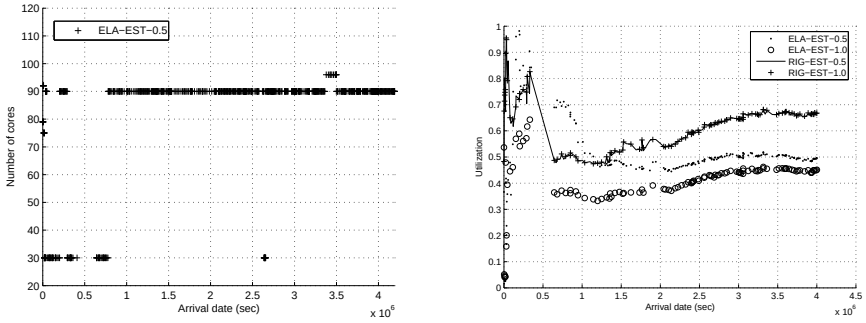

Fig. 9 Dynamics of the elasticity: (left) number of cores and (right) utilization.

ity 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, left graph, 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 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 5 (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.

Considering the queuing delay, the elastic scheme significantly improves the quality of service. For interactive jobs, the distribution of the queuing delay (Fig. 8, 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 4). Moreover, the impact of the settings (estimation vs oracle, and scaling factor λ_e) are 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 knowledge for $\lambda_e = 0.5$.

Fig. 9 provides information about the dynamics of the elasticity. The leftmost plot shows the evolution of the number of cores along time. Comparing with the service requests (Fig. 2), 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 actual site configuration (81 cores). The rightmost plot shows the cumulative 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 over the period is in the range 40%-50%, while the rigid utilization can reach a steady 70%. Significant over-provisioning seems thus to be required under moderate to high service requests for obtaining QoS. However, it must be stressed that this over-provisioning is compensated by the downsizing of the resource pool when the workload is low. In fact, the site utilization under PBS (not shown) is close to 50%, but as an average of lower utilization in the beginning, and higher in the period of high load. The overall cost (in units of cores) of the RL method is thus the same as the one of the physical site but with better service to the user.

6.4 Discussion

In some configurations, the optimization for interactive jobs is defeated, as revealed both by the threshold in the distribution of the responsiveness utility and unacceptable delays suffered by 5% of the jobs. Indeed, from Table 4, some interactive jobs actually experience very large delays (although in lower number than in the rigid MDP), up to more to 10^4 seconds, which explains the relatively large values of the mean. Since even the reference setting (ORA-1) keeps too many (slightly less than 4%) of the jobs above 2 minutes, the limits might be in the method itself. More precisely, the alternative is that either the exploration/exploitation parameter is to be increased, or the impact of the hypothesis that the machine pool can evolve only at a relatively low frequency (15 minutes of simulated time). This parameter may limit the capacity to adapt the resources to the bursts in the service request process (Fig. 2). The over-provisioning mentioned before could be attributed to the same cause. It would this be worth to experiment with a null delay, even if this setting is unrealistic. A complete simulation is not possible, due to the prohibitive cost of solving the continuous approximation problem (training the neural network). However, experiments on limited segments indicate that the limit is actually in the method: the models learnt in the burst phases has sufficiently long-term impact to limit the adaptation before a new burst occurs.

7 Related work

Jensen introduced time utilities functions, and models the scheduling objective as the maximization of the aggregated utility (Utility Accrual, UA) over time, for single processor [23] and later for multi-unit [39] real-time scheduling. The UA paradigm is a key concept in that it formalize the goal of optimizing the productivity of the system over time. The advances in RL, and more specifically in Temporal Difference Learning (TDM), have relatively recently allowed harnessing the UA paradigm with implicit prediction of the future.

Considering the application of RL to prioritization scheduling in mixed workloads, which is the target of this paper, this domain has been explored in depth by Tesauro, in the context of data centers [36,38]. 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. Our work considers a multi-objective optimization problem: the objectives (fairness and responsiveness, or utilization and

responsiveness) are fully heterogeneous, and the performance have to be evaluated separately. A second difference lies in the timescales, and thus the acceptable hypotheses. The data centers considered in Tesauro's work process 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 much less experience is available for grids, the first modeling studies show a much more complex behavior, at least multimodal (e.g. [25] models the inter-arrival time by a modulated Poisson process), and the workload itself is likely to exhibit long range memory [26]. As a consequence of these two differences, our variable selection is much less parsimonious than in Tesauro's work, because we try to represent the system by aggregated observed quantities (on the numerical side) rather than by model parameters, and also by categorical data (the VOs).

Scheduling exemplifies the need for extending RL from discrete to continuous value functions. Neural networks have proved experimentally to be efficient approximators of the value function in a purely Markovian context [37]. Convergence guarantees for such methods are rarely available, and realistic counterexamples do exist [6], in contrast with the proved convergence of many update strategies, including TDM, in the discrete case. 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 (exemplified in our work by the 15 minutes minimal delay for modifying the number of cores). Thus, introducing some memory of past states has been recognized as necessary, for instance in [38] when applying RL to a coupled scheduling and provisioning problem. Echo State Networks (ESN) were found to be particularly well suited for learning and predicting time series [22]. ESN are also relatively easy to train, compared to other recurrent networks. Moreover, some convergence results for RL based on ESN approximators have been recently obtained [35]. Thus, embedding them inside the learning process is a promising way to address problems where the past states and actions might be relevant. An alternative way to cope with value function approximation proposed by Vengerov [40] is learning the parameters of a fuzzy rule-base, with applications to scheduling, either of moldable jobs or of data transfers. However, this method implies an *a priori* parameterization of the scheduling problem, which lacks flexibility.

Our work defines the utility functions ab initio. In the Service Level Agreement (SLA) framework, they could be externally imposed. SLAs including universal terms (variables), and analytical combinations of them [32] exactly meet the requirement for expressing the high level objectives of users and administrators.

Finally, it may be useful to motivate the choice of the data-set from EGEE. Both the Grid Observatory [1,28] and the Grid Workloads Archive (GWA) [10] provide traces, albeit with different logic and goals. The available traces from the GWA are mostly oriented towards parallel workloads and/or co-allocation, which are not directly covered by our state-action model. Conversely, in the EGEE trace, the dependencies (in any) have been resolved before enqueuing, thus the model does apply. Furthermore, the analysis presented on the GWA site for the DAS-2 trace indicates that the workload characteristics would not offer much optimization opportunities, as system is somehow lightly loaded (average 10%, maximum 39%), and with an overwhelming majority of short or very short jobs (90% percentile below 10 minutes).

8 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, and calls for autonomic methods. The scope of the work presented here covers two real grid scheduling situations. In the matchmaking case, the grid workload is first dispatched to distributed sites, where actual scheduling happens; we have shown that RL is a good candidate for this level. The method is directly applicable to overlay or traditional schedulers, which feature a centralized job pool.

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.

Acknowledgments

This work was partially funded by the French national research agency (ANR), NeuroLog project (ANR-06-TLOG-024), by the EGEE-III EU project INFSO-RI-222667, and by DIM-LSC DIGITEO contract 2008-17D.

References

1. The grid observatory portal. www.grid-observatory.org.
2. L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *12th Int. Conf. on Machine Learning*, pages 30–37. Morgan Kaufmann, 1995.
3. P. Beckman, S. Nadella, N. Trebon, and I. Beschastnikh. SPRUCE: A System for Supporting Urgent High-Performance Computing. *IFIP series*, (239):295–311, 2007.
4. C. Blanchet, C. Combet, and G. Deleage. Integrating bioinformatics resources on the egee grid platform. In *6th ACM/IEEE Int. Symp. on Cluster Computing and the Grid*, page 48, 2006.
5. 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.
6. 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. MIT Press, 1995.
7. D.J. Colling and A.S McGough. The GRIDCC Project. In *1st Int. Conf. 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. D. Weissenbach E. Clévédé and B. Gotab. Distributed jobs on EGEE Grid infrastructure for an Earth science application: moment tensor computation at the centroid of an earthquake. *Earth Science Informatics*, 7(1-2):97–106, 2009.
10. A. Iosup et al. The Grid Workloads Archive. *Future Gener. Comput. Syst.*, 24(7):672–686, 2008.
11. C. Germain-Renaud et al. Grid analysis of radiological data. In Mario Cannataro, editor, *Handbook of Research on Computational Grid Technologies for Life Sciences, Biomedicine and Healthcare*. IGI Press, 2009.
12. E. Laure et al. Programming the Grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.

13. F. Gagliardi et. al. Building an Infrastructure for scientific Grid computing: status and goals of the EGEE project. *Phil. Trans. of the Royal Society A*, 1833, 2005.
14. J. Montagnat et al. Workflow-Based Data Parallel Applications on the EGEE Production Grid Infrastructure. *Journal of Grid Computing*, 6(4):369–383, 2008.
15. 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.
16. 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.
17. C. Germain-Renaud, C. Loomis, J. Moscicki, and R. Texier. Scheduling for responsive grids. *Journal of Grid Computing*, 6(1):15–27, 2008.
18. 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.
19. C. Germain-Renaud and O. F. Rana. The Convergence of Clouds, Grids, and Autonomics. *Internet Computing*, 13(6):9, 2009.
20. G. J. Gordon. Reinforcement learning with function approximation converges to a region. In *Advances in Neural Information Processing Systems*, pages 1040–1046. The MIT Press, 2001.
21. H. Jaeger. Adaptive nonlinear system identification with Echo State Networks. In *Advances in Neural Information Processing Systems 15*, pages 593–600. MIT Press, 2003.
22. H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78 – 80, 2004.
23. E. Douglas Jensen, C. Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *IEEE Real-Time Systems Symposium*, pages 112–122, 1985.
24. L.Amar, A. Barak, E. Levy, and M. Okun. An on-line algorithm for fair-share node allocations in a cluster. In *7th IEEE/ACM Int. Symp. on Cluster Computing and the Grid*, pages 83–91, 2007.
25. H. Li and M. Muskulus. Analysis and modeling of job arrivals in a production grid. *SIGMETRICS Perform. Eval. Rev.*, 34(4):59–70, 2007.
26. H. Li, M. Muskulus, and L. Wolters. Modeling correlated workloads by combining model based clustering and a localized sampling algorithm. In *21st Int. Conf. on Supercomputing*, pages 64–72, 2007.
27. I.M. Llorente. Researching Cloud Resource Management and Use: The StratusLab Initiative, 2009. The Cloud Computing Journal. http://cloudcomputing.syscon.com/node/856815.
28. C. Loomis. The Grid Observatory. In *Grids Meet Autonomic Computing workshop at ICAC'09*. ACM, 2009.
29. A. Luckow, L. Lacinski, and S. Jha. SAGA BigJob: An Extensible and Interoperable Pilot-Job Abstraction for Distributed Applications and Systems. In *10th ACM/IEEE Int. Symp. on Cluster, Cloud and Grid Computing*, 2010.
30. J. Perez, C. Germain Renaud, B. Kégl, and C. Loomis. Utility-based Reinforcement Learning for Reactive Grids. In *5th IEEE International Conference on Autonomic Computing*, 2008. Short paper.
31. C. E. Rasmusen and C. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.
32. R. Sakellariou and V. Yarmolenko. Job Scheduling on the Grid: Towards SLA-Based Scheduling. In L. Grandinetti, editor, *High Performance Computing and Grids in Action*. IOS Press, 2008.
33. Q. Snell, M.J. Clement, D.B. Jackson, and C. Gregory. The Performance Impact of Advance Reservation Meta-scheduling. In *IPDPS '00/JSSPP '00*, pages 137–153. Springer-Verlag, 2000.
34. R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.
35. I. Szita, V. Gyenes, and A. Lorincz. Reinforcement Learning with Echo State Networks. In *Artificial Neural Networks, ICANN 2006*, pages 830–839. Springer, 2006.
36. 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.
37. G. Tesauro and T.J Sejnowski. A parallel network that learns to play backgammon. *Artificial Intelligence*, 39(3):357–390, 1989.

38. G. J. Tesauro, W. E. Walsh, and J. O. Kephart. Utility-Function-Driven Resource Allocation in Autonomic Systems. In *Int. Conf. Autonomic computing and Communications*, pages 342–343, 2005.
39. H. Wu, B. Ravindran, E. D. Jensen, and U. Balli. Utility accrual scheduling under arbitrary time/utility functions and multiunit resource constraints. In *IEEE Real-Time and Embedded Computing Systems and Applications*, pages 80–98, 2004.
40. D. Vengerov. A reinforcement learning approach to dynamic resource allocation. *Eng. Appl. Artif. Intell.*,20(3), 2007.