Grid Differentiated Services: a Reinforcement Learning Approach

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

Grid Differentiated Services: a Reinforcement Learning Approach

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

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

Balazs 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—**Large scale production grids are a major case for autonomic computing. Following the classical definition of Kephart, an autonomic computing system should optimize its own behavior in accordance with high level guidance from humans. This central tenet of this paper is that the combination of utility functions and reinforcement learning (RL) can provide a general and efficient method for dynamically allocating grid resources in order to optimize the satisfaction of both end-users and participating institutions. The flexibility of an RL-based system allows to model the state of the grid, the jobs to be scheduled, and the high-level objectives of the various actors on the grid. RL-based scheduling 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. Our experimental results, both on a synthetic workload and a real trace, show that RL is not only a realistic alternative to empirical scheduler design, but is able to outperform them.**

I. INTRODUCTION

Large scale production grids are a major case for *autonomic computing*. Following the definition of Kephart, [4], an autonomic computing system should optimize its own behavior in accordance with high level guidance from humans. This central tenet of this paper is that the combination of utility functions and reinforcement learning can provide a general and efficient method for dynamically allocating grid resources to optimize the satisfaction of both end-users and participating institutions.

The exponential increases in network performance and storage capacity [8], together with ambitious national and international efforts, have already enabled the virtualization and pooling of processors and storage in advanced and relatively stable systems. However, it is more and more evident that the exploitation model for these grids is somehow lagging behind. At a time where industry acknowledges interactivity as a critical requirement for enlarging the scope of high performance computing [6], grids cannot anymore be envisioned only as very large computing centers providing batch-oriented access

to complex scientific applications with high job throughput as the primary performance metric.

A much larger range of grid usage scenarios is possible. Seamless integration of the grid power into everyday use calls for unplanned and interactive access to grid resources. A critical issue for widespread adoption of grids is thus to provide *differentiated* quality of service (QoS), covering the whole range from interactive usage with turnaround time as the primary performance metric, to the traditional batch-oriented usage [2].

Virtual Organizations (VO's) are the second key concept in the grid exploitation model: they represent groups of users with similar access rights. A VO generally matches a scientific community, which has institutional counterparts. Each institution contributes to the grid, by making its computing resources available, and by maintaining them. Thus, each VO is entitled to a pre-defined share of the resources, defined by agreements between the participating institutions. However, this agreement is also at the institutional level: it should be enforced in the mid- to long- time scale.

Applying the autonomic programme to job scheduling leads to the following constraints. First, high-level goals, such as QoS on one hand, and fair-share on the other hand, should be exposed by the scheduling system and should be easily tunable by respectively users and system administrators. Second, grid computing infrastructures are heterogeneous, dynamic, non steady-state systems, with only partial perception of their environment. Third, a production grid continuously provides a large sample of the input space (the jobs to be scheduled), allowing for statistical methods.

For these reasons, grid scheduling has been formalized as a reinforcement learning (RL) problem. The flexibility of an RL-based system allows to model the state of the grid, the jobs to be scheduled, and the high-level objectives of the various actors on the grid. RL-based scheduling 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 environ-

ment, including user requests and infrastructure. Finally, the very fundamental idea of RL is looking for maximizing the expectation of the long-term benefits.

The rest of this paper is organized as follows. Section II analyzes the requirements for QoS in production grids. The reference architecture for this analysis is the EGEE grid [1]. While this is a very useful starting point for making realistic assumptions, we must strongly stress that our results do not depend on the specificities of the EGEE architecture. The goal of this section is to informally state what are our grid model on one hand, and the optimization problem on the other hand. Section III formalizes our scheduling problem as a Markov Decision Problem. Section IV describes the utility functions that express the long-term objectives of end users and administrators. In this section, we propose some examples of utility functions, in particular with the use of automatically derived time-utilities functions, but this is not limitative, and could be extended directly to collections of independent tasks. Sections V and VI report on the experimental setup and evaluations, before the usual conclusion and perspectives. Related work is discussed trough the paper.

II. SCHEDULING FOR PRODUCTION GRIDS

A. EGEE scheduling

EGEE (Enabling Grid for E-sciencE) features 41,000 CPU's distributed on 240 sites in 45 countries, and maintains 100,000 concurrent jobs for a large variety of e-Science applications. We first briefly describe the scheduling process enacted by the EGEE middleware (gLite), as an example of the general issue encountered in the production framework: major architectural choices do not only depend an engineering or scientific criteria, but also of sociological, administrative and institutional constraints.

The important consequence is that decision-making (human or automatic) is non only distributed, but largely independent: each participating site configures, runs, and maintains a batch system containing its computational resources. The scheduling policy for each site is defined by the local site administrator, and the overall scheduling policy evolves implicitly as the resultant of the local policies.

gLite integrates the sites' computing resources through a set of middleware-level services (the Workload Management System, the WMS), which accepts jobs from users and dispatches them to computational resources based on the users requirements on one hand, and the characteristics (*e.g.* hardware, software, localization) and state of the resources on the other hand. The WM is implemented as a distributed set of *resource brokers*, with some tens of them currently installed; all the brokers get an approximately consistent view of the resource availability through the grid information system. Each broker reaches a decision of which resource should be used by a matchmaking process between submission requests and available resources. Once a job is dispatched, the broker only reschedules it if it failed; it does not reschedule jobs based on the changing state of the resources.

Job requirements are exposed to the various services of the WMS via the Job Description Language (JDL), derived from the Condor ClassAd language. For instance, a job can expose its requirement for interactivity with the SDJ (Short Deadline Job) tag,.

B. Differentiated Quality of Service

Most sites on the EGEE grid infrastructure have implemented scheduling policies that, to first-order, execute jobs in a first-in, first-out (FIFO) order. On a large infrastructure, this provides reasonable scheduling latencies and execution times for workloads consisting of numerous, long-running tasks. On the contrary, this does not provide a reasonable QoS for most demanding applications coming from an increasingly diverse user community. For example, the QoS is inadequate for workloads that have a few, urgent tasks or that have many short tasks. To provide differentiated QoS for these applications, EGEE has experimented with specialized site configurations.

One class of applications requires a pseudo-interactive response from the grid scheduling. The spontaneous, interactive nature of these applications precludes using standard advanced reservations. Nonetheless, the Virtual Reservations scheme proposed in [2] do play an important role in the *Short-Deadline Job* configuration. This configuration guarantees that the job will either immediately start executing or be rejected it if no resource is available.

Another interesting case involves different, relative priorities between several applications within the same Virtual Organization (VO). Examples include favoring analysis jobs, debugging jobs and the like over more numerous, long-running simulation jobs. Two solutions have been shown to work on the EGEE infrastructure: 1) overlay task-management systems (e.g. DIRAC) and 2) implementing standardized fair-share policies on the sites. DIRAC can provide arbitrarily fine-grained policies to control the priorities, but all tasks must be submitted through a centralized meta-scheduler. The other solution allows a range of different submission scenarios, but provides only coarse-grained priorities, requires complex configurations at the site level, and fragments the resource usage. Finally, both of these techniques provide statistical guarantee of fast scheduling of high-priority tasks only for VOs with access to a large number of resources.

C. Architecture

We assume that the scheduling is the resultant of two successive steps.

- matchmaking: the incoming job is immediately dispatched onto the queue associated to a set of resources; the information about eligible resources and expected performance is available though a global information system.
- local scheduling: the job is dispatched on computing resources (machines); the information required to perform the scheduling decision is only local.

Our goal in this paper is to develop a scheduler for the site level, which is experimentally (at least in the EGEE case) the most difficult to adjust to the high-level requirements.

At the site level, we assume sequential jobs. It has been shown that utility functions can be derived from the DAG structure of parallel jobs [3]; thus this assumption can be relaxed in future work while keeping the same framework.

The long-term expected utilities defined by the local RL-based schedulers can be efficiently exploited by a distributed matchmaking processes that dispatch jobs to the site. These processes do not have to be privy to details of how the individual site optimize their resource allocation. The site can summarize its internal state by registering a site-level utility function that specifies the performance (utility) of receiving each possible categories of jobs, along QoS classes and VO. The matchmaking processes can then select a site for the incoming job by simple ranking the utility indices. A most ambitious scheme would implement a second level of RL, by integrating the site utilities with other information, for instance its knowledge about the site reliability, and the possible compound structure of the job.

III. THE RL FRAMEWORK

A. Markov decision process and reinforcement learning

We first give the mathematical formalization of decision making. A Markov decision process is a quadruple (S, A, P, R). S is the set of possible states the system can be in. A is the set of decisions that can be taken. P is a collection of transition probabilities

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

that probabilistically map the current state and action to the next state. and a reward function

$$R_{ss'}^a : S \times A(s) \times S^+ \to \mathbb{R}$$

with $A(s)$ is the set of possible actions for a state s and S^+ is the set of accessible states from S.

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. The objective function to maximize is the *value function*, which is the long-term expectation of the rewards, with a discount factor dampening the furthest rewards:

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

where $\gamma \in [0, 1]$.

In the scheduling context, the environment dynamics, P and R, are unknown. Thus, the Q function has to approximated through repeated experiments. This is the definition of reinforcement learning [9]: the optimal policy will be learnt by interactions with the environment. The general algorithm is an iterative process known as temporal-difference learning.

The policy learning framework used in this work is based on SARSA, a classical reinforcement learning algorithm. SARSA is

Initialize $Q(s, a)$ arbitrarily for all s and a
$s_0 \leftarrow$ current system state; Choose a_0 from s_0 from $\tilde{\pi}$
$s \leftarrow s_0$; $a \leftarrow a_0$
repeat
    Take action a; observe r and s'; choose a' from $\tilde{\pi}$
    $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$
    $s \leftarrow s'$; $a \leftarrow a'$
    until shutdown

Fig. 1. The SARSA algorithm

an *on-policy* learning algorithm (fig. 1): the approximate value function guides the selection of the current action a, thus the reward r and the future state s'. The $\tilde{\pi}$ policy is defined from the current approximation Q. More precisely, if $a*$ is the action which maximizes the expected reward considering the current approximation Q ($a* = argmaxQ(s', .)$), then $a*$ is often selected, but not always. Indeed, the algorithm must maintain a trade-off between exploitation (using the knowledge gained so far) and exploration (looking for potentially better actions). In this work, we have used the classical ϵ-greedy strategy: $a*$ (the best action) is chosen with probability $1 - \epsilon$; the choice between all other accessible actions is uniform.

B. Grid scheduling and the reinforcement learning paradigm

As explained before, a reinforcement learning formalization needs to define states, actions and rewards for the given task of optimal policy learning. A first contribution of our work is the proposition of a set of variables describing states and actions to allow the formulation of the grid scheduling problem as a continuous action-state space reinforcement learning problem. 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 limited set of real-valued variables.

- the expected time remaining until any of the currently running jobs is completed;
- the number of currently idle machines;
- the workload (the total execution time of jobs waiting in the queues);
- the average user-utility (see below) expected to be received by the currently running jobs;
- the current share of resources resulting from previous allocation, along each VO.

ACTION SPACE: THE JOB MODEL. 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 (extracted for instance from the EGEE logging and bookkeeping system). The exact set of variables is under research, for the time being we are using 1) the type of the job (batch/interactive), 2) the VO of the user who submitted the job, and 3) the

expected execution time, which is the time to complete the job without any queuing or management overhead. The first two descriptors are actually available; the third one can be estimated from other descriptors.

REWARD: UTILITY FUNCTIONS. The overall utility of the scheduler is a combination of the *time-utility*, and the *fairness*. The time-utility function [3], [11], [12] is attached to each job, and it describes how "satisfied" the user will be if his/her job finishes after a certain time delay. It is typically a decreasing function of time, and it can vary with the job type. The fairness represents the difference between the actual resource allocation and the externally defined shares given to VO's. These utility functions are described in more details in section IV-A.

C. Continuous state-action space

The state-action space is continuous (real valued). As a consequence, implementing the assignment $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)]$ 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 4 VOs, the state-actions pace is $\mathbb{R}^{11}$. The table representation has extremely poor scalability, resulting in too severe approximations. The alternative is to use a non-linear continuous approximation, as proposed in [10]. The design choice then lies in the interpolation method , e.g. neural networks (NN), Gaussian processes [7]), or kernel methods, to cite a few of the available classes of algorithms. Because at this step there is no prior knowledge on the properties of the value function Q, we opted for NN, which are both flexible and easily available off-the-shelf.

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 NN, there are two possibilities: active learning, where the newly acquired value is used as a new training example, and re-learning, where the set of examples is enriched with the new one, and the NN is trained from scratch with the new set. While active learning is a very active research area, the simple procedure outlined above has shown to be deceptive, thus the second one has been preferred here. More precisely, only a sliding window on the past is kept, in order to limit the size of the training set.

As pointed in [10], there are two consequences of this combination of algorithms. The first one is that the exact values of the value function Q are not available, because the learning step is stochastic. Thus the modified SARSA computes a noisy approximation (termed regression in the Machine Learning framework) of Q. The effectiveness of the approximation must thus be tested against the actually observed utility. The second consequence is that there is no theoretical guarantee that the overall algorithm does converge, even if separately, the discrete (or the abstract continuous) SARSA algorithm and the NN regression are proved to converge. The reason is that at each step, the target function of the NN actually changes with the observed rewards and state. Thus the convergence of the algorithm has to be checked experimentally.

The final ingredient in the definition of the algorithm is the initialization. In a very complex optimization landscape, running the modified SARSA algorithm with an untrained NN 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 second because of a poor initial approximation of the value function. To overcome this initialization issue, the RL system is trained off-line with an m̃ early deadline first policy $\tilde{\pi}$. After a few learning sweeps using collected rewards, the network is quickly usable to take its own decisions and be optimized throughout the real rewards.

IV. THE UTILITY MODEL

A. Job utility functions

Jensen at al. [3] introduced the concept of time utilities function (TUF). TUF provide a unified framework for describing various QoS requirements, including best effort, hard real-time and soft real-time. In general, the TUF of a job is any function of time t which defines the user-perceived utility of completing a job at time t. In the most elementary setting, the TUF of a batch job is constant; the TUF of a hard real-time job is stepwise: up to the deadline L, the utility is constant, and becomes null after the deadline. The TUF associated to soft-real time is constant up to the deadline, and decreases rapidly after.

However, these simple utility functions 1) fail to capture the evident fact that a batch job must return in reasonable time, and 2) require a definition of deadline and the decrease function for the real time jobs. In order to make a step towards self-configuration, the TUF should be derived in a semi-automatic fashion. We propose the following scheme for self-defined TUF (fig. 2). Let τ_j be the execution time of job j (here and in the following, job-related quantities are indexed by j, in order to contrast them with the constants).

- The relative deadline d_j (*i.e.* the absolute time deadline minus the submitting time a_j) is the execution time plus a fixed startup time σ: $d_j = \tau_j + \sigma$. Indeed, even extremely short jobs cannot expect to be completed instantaneously; σ captures the overhead associated with traversing the various middleware services, before the job is dispatched on a site and starts waiting for available resources.
- The user should provide an indication of the QoS requirement associated to the job. In this work, we consider only a binary choice, between interactive and batch jobs.
- For batch jobs, the utility decreases over time following a power law with exponent β.
- For interactive jobs, the utility decreases exponentially over time, at rate α.

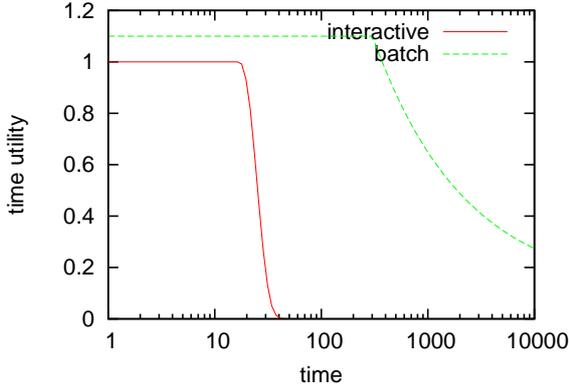Thus, if U_j^B is the batch utility and U_j^I is the interactive one, we get:

Fig. 2. Self-defined time-utility functions

$$U_j^B(t) = U_j^I(t) = 1 \quad \text{if} \quad a_j \le t \le a_j + d_j, \quad (1)$$

$$U_j^I(t) = e^{-\alpha(t - a_j - d_j)} \quad \text{if} \quad t > a_j + d_j, \quad (2)$$

$$U_j^B(t) = \left(\frac{t - a_j}{d_j}\right)^{-\beta} \quad \text{if} \quad t > a_j + d_j. \quad (3)$$

The utility can receive an interpretation as a target probability (in the frequentist sense). We would like to have the following behaviour: in the long run, the probability that the actual turnaround time of a job is larger than a positive value t is the utility of the job at time t.

Another important point is that these utility functions allow to define α and β in a way that is consistent with the high-level requirements of interactive and batch jobs. Consider $u_{1/2}$, the value of t for which the utility is 0.5 (half the maximal utility). In the interactive case, we get $u_{1/2} = a_j + d_j + \log(2)/\alpha$, which shows that the user satisfaction depend on the wall-clock waiting time. In the batch case, the corresponding equation is $u_{1/2} - a_j = 2^{-\beta} d_j$; the penalty is roughly proportional to the execution time, because the relative deadline d_j is the execution time augmented by the overhead, which should be negligible for batch jobs. Thus the shape of the utility curve for batch jobs scales with the job size, while the shape of the utility curve for interactive jobs is fixed by external requirements. Within this framework, it is obviously possible to define multiple classes of service, by varying the α and β parameters.

B. Fairness and productivity

The allocation process should be such that the service received by each VO is proportional to some share. If there are n VOs, the shares are usually expressed as a n-vector of the percentages of the total resources $w = (w_1, \ldots w_n)$. As stated before, these shares are a priori parameters of the scheduling problem. Thus, contrary to the previous section, the modeling step should only address the following issue: define a function of the service actually received which is maximal when the proportionality is perfectly achieved.

Let $S_k(t)$ be the fraction of the total service received by VO k up to time t. Then, the deficit distance between the

optimal allocation, and the actual one, is a good measure of the unfairness. The deficit distance is defined as

$$D = \max_k (w_k - S_k)_+,$$

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

The unfairness is bounded above. A fairness utility can thus be derived by a simple linear transform. If M is the maximal unfairness, the fairness utility F is

$$F = -\frac{D}{M} + 1. \quad (4)$$

Some VO 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 [5]), 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 for favoring the user utility in the short term. Thus we keep the fairness utility as defined in eq. 4.

V. EXPERIMENTAL SETUP

A. The simulation platform

We developed a simulation framework for learning and evaluation of grid scheduling policies. This discrete events simulator supports multiple queues, fair-share measurement, multiple types of jobs and independent definition of the scheduling policy. The RL scheduler features the modified SARSA algorithm and the implementation of various utility functions. As a comparison baseline, we also have implemented a FIFO scheduler. Both the simulator and the schedulers are developed with MATLAB.

In the reported experiments, we have used the following parameters values for the utility functions: $\alpha = 0.5, \beta = 0.3$. With these values, an interactive jobs is down to 0.5 (ie half of the maximum utility) 1.3 units of time after the deadline, and the utility of a batch job is down to 0.5 when the turnaround time is approximately 10 times the execution time, meaning that the waiting time is 9 times the execution time.

σ, the startup time, is 1 minute, consistent with experimental data on production grids.

Considering the modified SARSA, the parameters are, on one hand, the ϵ parameter regulating the exploration-exploitation trade-off, and on the other hand the neural network parameters. Here, $\epsilon = 0.3$; the neural network is a standard multi-layer perceptron with one hidden layer containing 20 sigmoidal hidden units; the back-propagation learning rate is 0.3; the SARSA discount parameter γ is set to 0.2.

B. The workloads

We analyze two workloads. The first one is the traditional M/M/1 queue, and the second one is extracted from real EGEE traces.

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. The definition of interactive jobs is set to jobs with an execution time less than 15 mn, and the proportion of interactive jobs in the overall workload is set to a fixed number, 20%. The value of mu follows immediately, giving an average execution time of approximately 67 minutes. 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.

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 work in the overall synthetic workload are the same as the target ones.

The EGEE workload: This experiment uses as input a trace of real EGEE jobs. The trace covers the activity of more than one week (17-25 May 2006) at the LAL site, and includes 5000 user jobs, not counting he monitoring jobs, which are executed concurrently with the users jobs, and consume virtually no resource, which have been removed from the trace.

This particular segment has been selected in a one year trace, for the following reasons.

- A constant number of processors ($P = 100$). The site has been restructured many times in the whole extent of the trace, increasing its resources from 25 to 400 processors.
- The requested load is nearly conformant to the target fairshare weights, which are 0.2, 0. 12, 0.12, 0.6, 0.6, 0.9, 0.35. The six first VOs are real ones, and the last weight is the aggregation of the remaining "small" VOs entitled to access the site.
- The load is significant: the overall utilization is 0.46.

C. Performance metrics

The first question is the execution time of the RL algorithm itself, that is the time to take a scheduling decision. Within our MATLAB platform, the average execution time of the RL algorithm ranges from 1 to 10 ms, depending on the load. Indeed, the RL scheduler has to scan the waiting jobs in order to select the one maximizing the reward, thus the execution time depends on the system load. Obviously, a real-world scheduler would not be matlab-based, but these figures show that the RL scheduler is realistic.

The most important performance indicators are related to the satisfaction of the grid actors. From the user point of view, we consider the *relative overhead*, which is the ratio of the waiting time to the execution time. For the faire-share point of view, the normalized distance to the optimal .
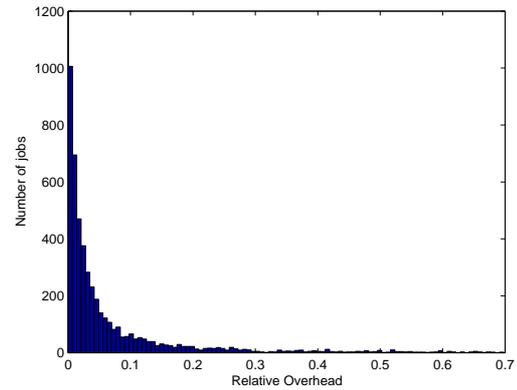
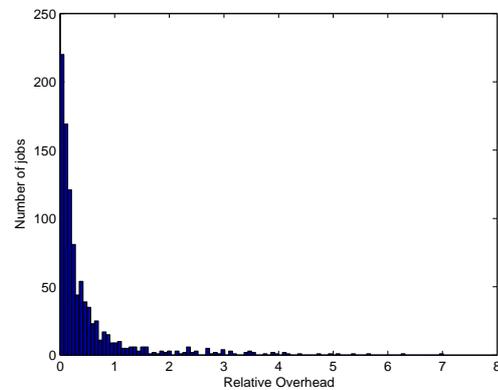Fig. 3.   Distribution of the relative overhead under RL -All jobs

Fig. 4.   Distribution of the relative overhead under RL - Interactive jobs

VI. PERFORMANCE RESULTS

A. The synthetic workload

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. With this parameter, the simulated time range is 111 hours, thus more than 4 days.

The distribution of the relative overhead is shown on fig. 3
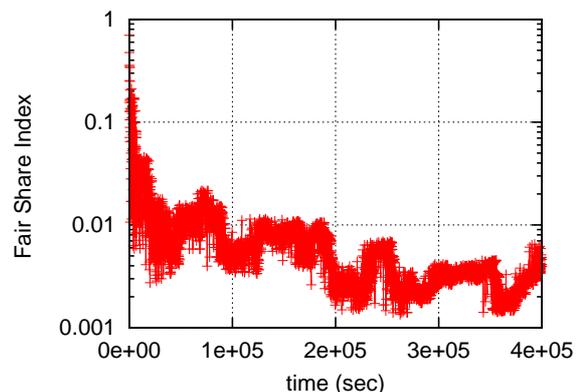
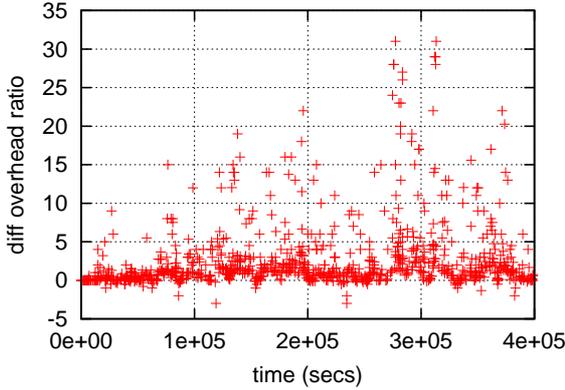Fig. 5.   Normalized distance to the target fair-share

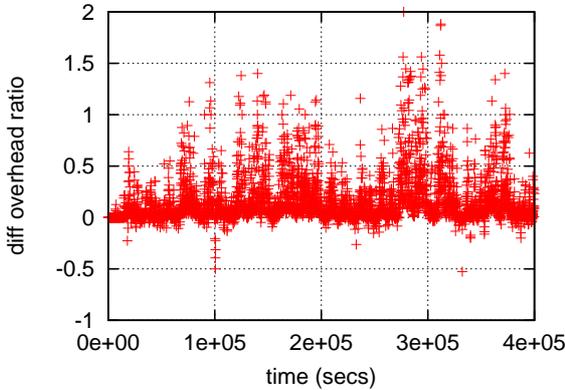Fig. 6. Difference between the relative overheads under FIFO and under RL - Interactive jobs

Fig. 7. Difference between the relative overheads under FIFO and under RL - Batch jobs

Fig. 8. Distribution of execution time in the trace

for all jobs, and on fig. 4 for interactive jobs. The results are quite good: for the overall workload, all overheads are below 1; for the interactive workload, where the relative overhead is naturally larger (because the execution time is smaller), more than 97% of the jobs incur an overhead less than 0.5.

Fig. 5 shows the fair-share performance. With a feasible schedule, in the long run, the job sample is conformant to the target, thus any scheduler achieves the requested fair-share at the end of the simulation. The question is thus to see if the RL method is able to integrate quickly enough the fair-share goal. After time 1E5, the normalized distance to the target is very small, in the order of 0.01 or below. As can be expected, the RL algorithm takes some time to stabilize. However, this time is fully compatible with the grid scale, slightly more than one day.

The next step is to compare the performance of our method with a baseline one. The baseline in this experiment is the FIFO scheduling. Fig. 6 and 7 show the difference between the overhead ratio of the FIFO method and the RL one, respectively for interactive jobs and batch ones (because we measure overhead, smaller is better, thus a positive difference indicates that RL is better). In both cases, RL almost always outperforms FIFO, and by a large factor in the interactive case. The average overhead ratio for interactive jobs under FIFO
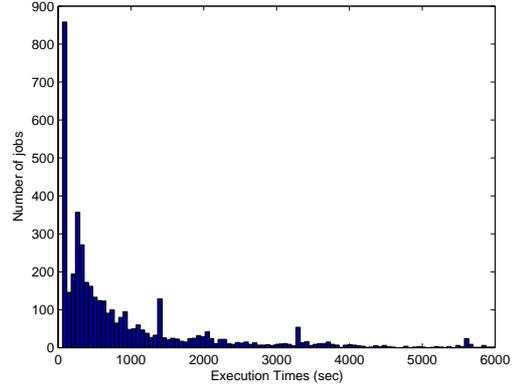
scheduling is 3.19, while it is 0.52 under RL scheduling. Batch jobs are also better scheduled, with respect to their deadlines, with the RL method. The average overhead ratio for batch jobs under FIFO scheduling is 0.2, while it is 0.04 under RL scheduling. This better performance has a cost: many batch jobs incur a small penalty with respect to FIFO, as shown in fig. 7.

B. The EGEE workload

Figure 8 shows the distribution of the workload, which is obviously much more complicated than the synthetic one. In fact, the distribution is only a partial one: the 600 longest jobs have been removed (of the histogram, but not the experiment), in order to get a viewable histogram. We are currently analyzing the full trace (one year), and it is very likely that the workload distribution is heavy-tailed. The workload is heavily dominated by short jobs. This characteristic is by no way specific of the selected trace, but a general feature of a significant part of the EGEE workload [2]. In order to keep our previous fraction of 20% of interactive jobs, the jobs with execution time less than 200 seconds are considered as interactive ones.

The real site scheduler is MAUI/PBS, with the SDJ mechanism partially enabled. As explained in section II-B, the accepted SDJ jobs are executed immediately, within reserved slots. The challenge for the RL algorithm is thus to be able to compete with this mechanism without prior reservation.

The distribution of the relative overhead for the RL method is shown on fig. 9 for all jobs, and on fig. 10 for interactive jobs. On average, the RL scheduler is only marginally better than the real scheduler for the overall workload (11.3 for the RL, and 13.5 for the real), and significantly worse for the interactive workload (36.4 vs 18.7).

The problem here is the learning period. Fig 11 shows the average difference between the relative overhead under the real scheduler and under the RL one, computed from the current time to the end of the simulation. After time 4.5E5, the RL scheduler behaves consistently better than the real one, and is continuously improving. Thus, the RL method is in fact
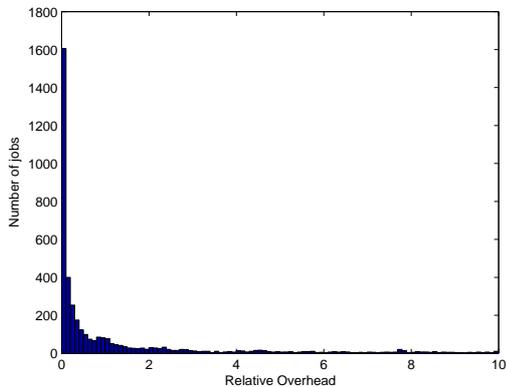
Fig. 9. Distribution of the relative overhead under RL scheduling - All jobs
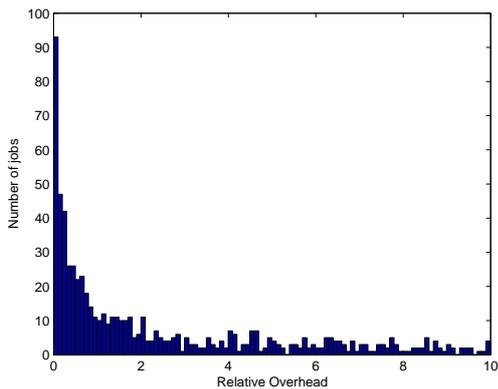

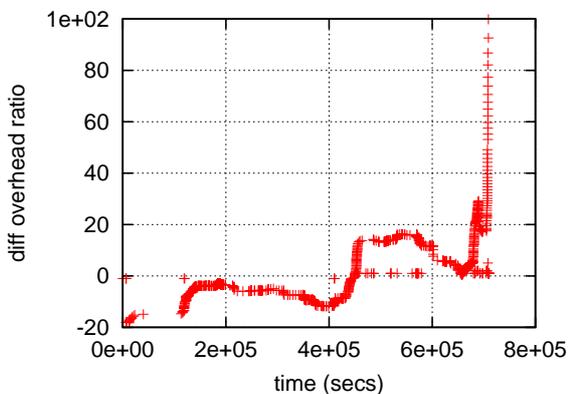Fig. 10. Distribution of the relative overhead under RL scheduling - Interactive jobs

Fig. 11. Average difference between overhead ratio under the real scheduler and RL scheduling - Interactive jobs
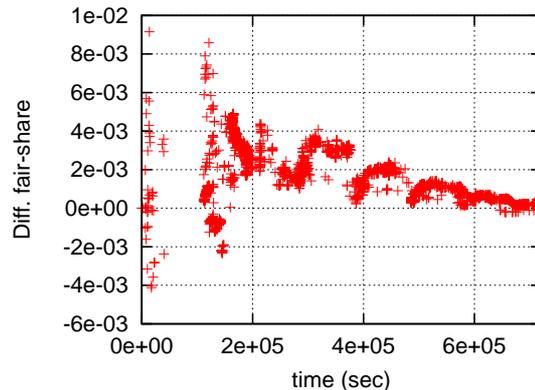

Fig. 12. Difference in fair-share under RL and the real scheduler

able to outperform the real scheduler, but the learning phase is much longer than in the synthetic workload. The conclusion will sketch the paths to speedup the learning phase.

Figure 12 shows the difference in the fair-share under the RL scheduler and the real one. The RL scheduler achieves nearly the same performance, the difference being constantly below 0.01.

VII. CONCLUSION

The main contribution of this paper is to present and demonstrate a general framework for providing both QoS and fair-share in an autonomic fashion. This framework is based on 1) configurable utility functions and 2) RL as a model-free policy enactor.

Combining RL methods and utility functions for resource allocation has been pioneered by Tesauro [11], [10] and Vengerov [12]. Tesauro's work targets optimal allocation of resources for Data Centers, thus optimizes the fraction of a global pool allocated to each application, while we are seeking an optimal schedule. Nevertheless, the resource allocation issues are very similar. The main difference in our work is that we consider a multi-criteria optimization problem, including a fair-share objective.

The comparison with a real and sophisticated scheduler shows that the most immediate improvement of our RL scheme should be researched in the learning phase. More sophisticated interpolation (or regression) could speedup this phase. More fundamentally, a hybrid scheme, as proposed in [10], where the RL is calibrated off-line by using the results of a real scheduler, should be explored. However, this raises a new issue: our trace show that the workload profile is subject to abrupt changes. While the RL is by construction able to deal with a slowly changing situation, the adaptation of an optimization scheme to major changes needs to define and explore of a landscape of profiles.

REFERENCES

[1] 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.

[2] C. Germain-Renaud, C. Loomis, J.T. Mo'scicki, and R. Texier. Scheduling for responsive grids. *Journal of Grid Computing*, 2007. on line doi= 10.1007/s10723-007-9086-4.

[3] 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.

[4] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[5] L.Amar, A. Barak, E. Levy, and M. Okun. An on-line algorithm for fair-share node allocations in a cluster. In *CCGRID*, pages 83–91, 2007.

[6] I. Mirman. Going Parallel the New Way. *Desktop Computing*, 10(11), June 2006

[7] Carl Edward Rasmusen and Chris Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] L. G. Roberts. Beyond Moore's law: Internet growth trends. *Computer*, 3(1):117–119, 2000.

[9] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

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

[11] 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.

[12] D. Vengerov. A reinforcement learning approach to dynamic resource allocation. *Eng. Appl. Artif. Intell.*, 20(3), 2007.