

# Resource Allocation in the Grid Using Reinforcement Learning

Aram Galstyan, Karl Czajkowski, and Kristina Lerman  
USC Information Sciences Institute  
4676 Admiralty Way, Marina del Rey, CA 90292-6695  
galstyan@isi.edu, karlcz@isi.edu, lerman@isi.edu

## Abstract

*In this paper we study a minimalist decentralized algorithm for resource allocation in a simplified Grid-like environment. We consider a system consisting of large number of heterogeneous reinforcement learning agents that share common resources for their computational needs. There is no communication between the agents: the only information that agents receive is the (expected) completion time of a job it submitted to a particular resource and which serves as a reinforcement signal for the agent. The results of our experiments suggest that reinforcement learning can be used to improve the quality of resource allocation in large scale heterogeneous system.*

## 1. Introduction

Grid computing is an emerging technology that enables users to share a large number of computing resources distributed over a network. The dynamic, *federating* nature of Grid policy environments is dominated by virtual organizations (VOs) which associate heterogeneous users and resource providers. It is not known how large individual VOs will be, but it is reasonable to imagine resource sharing among populations with tens of thousands of users and thousands of resources. Hence, allocation mechanisms need to be highly scalable and robust to localized failures in resources and communication paths. A further challenge to Grid resource allocation lies in the lack of accurate resource status information at the global scale. Hence, a feasible allocation mechanisms should not depend strongly on the availability of current global knowledge.

Although there has been considerable attention given to the resource allocation problem in the Grid [2], very few researchers have addressed the problem from the perspective of learning and adaptation. Meanwhile, the multi-agent systems (MAS) and distributed AI communities have shown that groups of *autonomous learning agents* can successfully solve different load balancing and resource allocation prob-

lems [3, 4]. The goal of this paper is to apply multi-agent learning techniques to the problem of resource allocation in the Grid. The MAS approach is well suited for describing the Grid, because the distributed, autonomous nature of agents (Grid users and resources) reflects the federated nature of the Grid. Introducing learning allows the multi-agent system to adapt to changes, such as the changing resource capacities, resource failure, or introduction of new agents into the system. Furthermore, we believe that the MAS approach will prove useful for policy design, because it can be used to study the performance of a VO implementing a given resource allocation strategy to verify that it does not lead to any unintended global consequences.

## 2. The Model

In real Grid applications the problem of mapping resources to specific jobs can be very complex, and may require co-allocation of different resources such as specific amount of CPU hours, system memory, network bandwidth for data transfer, etc. In this paper we neglect the need for co-allocation, and assume that jobs generated by a user require only certain CPU-time so that they are uniquely characterized by their duration.

**Resources Providers:** The scheduling decision for a contemporary batch system is too computationally expensive for us to perform thousands of times per time-step in our agent simulations. In our model, we consider a simplified representation of the resources and local schedulers. Namely, we assume that each resource is characterized by its capacity  $C$  which is defined as an inverse CPU time needed to complete a job of a unit length. Within this framework, there is only a single job running at the system at a given time (note that this approach is different from one adopted in Ref[4] where the capacity of the resource was assumed to be shared equally over all the jobs in the queue). For simplicity, we will assume that all the local schedulers prioritize the jobs by their arrival time (FCFS).

**Users:** We model users as heterogeneous selfish agents that try to maximize their utilities. Clearly, one can de-

fine agent's utilities in various ways. In this paper, we used weighted contributions of two metrics:  $\rho_i = a_i T_w + (1 - a_i) T_{exc}$ , where  $T_w$  is the queue wait time, and  $T_{exc}$  is the job execution time normalized to the duration of the job (i.e., inverse resource capacity), and  $a$  is a constant determining the weight of each contribution.

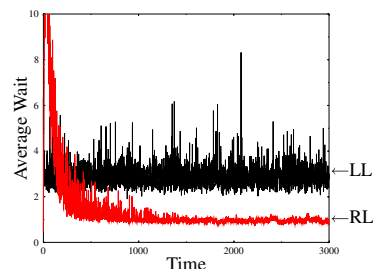
**Resource Selection:** To complete the definition of our model, we need to describe how the agents use reinforcement learning (RL) to select resources. In this paper we use  $Q$ -learning with  $\epsilon$ -greedy selection rule. For each possible action (i.e., selecting a specific resource) the agent keeps a  $Q$ -value that indicates the efficiency of that resource in the past. After each completed job, the agent gets a reinforcement signal (containing the start-time and the end time for that job), calculates the metric  $\rho_i$ , and translates it into a reward  $r$  for resource  $i$  that we have chosen as follows:  $r = \text{sign}(\langle \rho_i \rangle - \rho_i)$ , where  $\langle \cdot \rangle$  stands for averaging over all the submitted jobs. Finally, the agent updates the  $Q$  values according to  $Q_{i,t+1} \leftarrow Q_{i,t} + \alpha(r - Q_{i,t})$ , where  $\alpha$  is the learning rate.

To compare the performance of the  $RL$  algorithm we also studied two other resource selection rules: 1) *Random Selection (RS)*: Agents choose randomly with uniform probability between the resources. 2) *Least loaded (LL)*: In this model agents choose the least loaded resource to submit a job. To study the impact of crowding effect (where many agents choose the least loaded resource simultaneously) we introduce a parameter  $p$  so that once a job is submitted to a resource, the load of that resource is updated only with probability  $p$  (for the results presented in this paper we have used  $p = 1/4$ ).

### 3. Experimental Results

In this section we present the results of simulations for  $N = 1000$  agents and  $R = 250$  resources. At each time step, agents independently generate jobs at rate  $P$ , and the length of the jobs are taken randomly from the uniform distribution in the interval  $[J_{min}, J_{max}]$ . To take into account the wide dispersion in the job sizes in real Grid applications, we chose  $J_{min} = 10$  and  $J_{max} = 1000$ . The capacities of the resources were also chosen uniformly in the interval  $[C_{min}, C_{max}]$ , with  $C_{max} = 10$ ,  $C_{min} = 1000$ .

Our results indicate that for small value of job arrival rate the  $RS$  algorithm performs comparably or even better than both  $LL$  and  $RL$ —if the job load is sufficiently low, choosing resources randomly guarantees load balancing. The situation changes drastically as one increase the job arrival rate, however, and the performance of the  $RS$  algorithm deteriorates due to "bottlenecks". The  $RL$  algorithm, on the other hand, allows the agents to distribute jobs among the resources much more efficiently than the  $RS$  rule. More remarkably, we find out that for some parameter settings the



**Figure 1. Average wait time for  $LL$ (least loaded) and  $RL$  selection rules ( $P = 0.15$ )**

$RL$  algorithm performs quite well compared to the least loaded algorithm, as it is illustrated in Fig 1 where we plot the time evolution of the average job wait time. After a certain transient (learning) time the average wait for the  $RL$  algorithm falls well below wait time for the  $LL$  rule. Thus, although the agents do not exchange information nor have any global knowledge on the current load levels in the system, the learning mechanism allows them to efficiently distribute jobs among the resources.

### 4. Conclusion

The benefit we have observed for the  $RL$  algorithm over random selection already suggests an improvement over existing Grid metascheduling strategies, many of which, while performing substantial planning of job sequences etc., make random or otherwise uniform distribution decisions to spread work among several (or many) large-scale resources [1]. Even when metaschedulers attempt to use environmental information, such as load levels, our results suggest that the  $RL$  algorithm can provide better adaptive behavior because each metascheduler would learn from the environment's responses to its own queries<sup>1</sup>.

### References

- [1] B. Barish and R. Weiss. LIGO and the detection of gravitational waves. *Physics Today*, 52(10):44, 1999.
- [2] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A resource management architecture for metacomputing systems. In *JSSPP'98*, 1998.
- [3] A. Galstyan, S. Kolar, and K. Lerman. Resource Allocation Games with Changing Resource Capacities. In *AAMAS'03*, 2003.
- [4] A. Schaerf, Y. Shoham, and M. Tennenholtz. Adaptive load balancing: A study in multi-agent learning. *Journal of Artificial Intelligence Research*, 2:475–500, 1995.

<sup>1</sup> This research was supported by the DARPA under contracts number F30602-00-2-0573.