DeepScheduling: Grid Computing Job Scheduler based on Deep Reinforcement Learning

Lucas C. Casagrande, Guilherme P. Koslovski, Charles C. Miers, and Maurício A. Pillon

Abstract Grid systems are large-scale platforms which consume a considerable amount of energy. Several efficient resource/power management strategies were proposed by the specialized literature. However, most of the proposed strategies are rule-based policies which do not exploit workload patterns. Deploying the same set of rules on systems using different usage patterns, and platform settings, may lead to a sub-optimized setup. Due to the complex nature of grid systems, tailoring such a system-specific policy is not a straightforward task. In this paper, we explore a Deep Reinforcement Learning (DRL) method to build an adaptive energy-aware scheduling policy. We trained our algorithm using real workload traces from Grid'5000 platform. Our experiments pointed out an energy setup saving up to 7%, as well as average requests waiting time reduction of 27%. Finally, the results clarify the importance of explore the workload to build system-specific policies.

Key words: Deep Reinforcement Learning, Energy-Aware Scheduling, Grid

1 Introduction

Grid computing is an outstanding paradigm for provisioning geographical-distributed computing, store, and networking to compute-intensive applications. Specifically, Virtual Organization (VO) groups can be composed of different resource providers establishing a sharing relationship which will rule the discovery, allocation, and sharing of its local resources. Grid platform sizes can vary its computing resources from a few hundred to thousands, through incorporating more providers to its VO [21].

Lucas C. Casagrande · Guilherme P. Koslovski · Charles C. Miers · Maurício A. Pillon
Graduate Program in Applied Computing, Santa Catarina State University, Joinville, Brazil
lucas.casagrande@edu.udesc.br, {guilherme.koslovski, charles.miers, mauricio.pillon}@udesc.br

Sharing resources to compose large-scale platforms is an appealing choice to increase the computing power available with minimal capital investments, allowing users to explore deeper levels of parallelism to handle complex problems. However, the sharing complexity and operational costs increase proportionally [7]. Computing resources consumes considerable amounts of energy when powered on, and this consumption increases almost linearly to resource utilization. Considering the size of a grid platform, the energy consumption can become prohibitive, compromising sustainable aspects of grid platforms if not properly tackled [5, 8].

Optimizing energy does not involve only the choice of the most power-efficient hardware but it is also intrinsically related to the resource management strategy. At the platform level, shutdown strategies exploit the times between jobs to minimize the number of idle nodes by turning them off, while at the job level (user's perspective), Dynamic Voltage Frequency Scaling (DVFS) strategies exploit the times when jobs are not executing computation-intensive tasks by scaling down the processor frequency [15]. The complex nature of grid systems motivated the development of simple and scalable rule-based policies which usually do not take into account the job submission patterns. Furthermore, using the same set of rules on systems with different infrastructures and usage patterns leads to an under-optimized setup [11]. For example, if the inter-arrival time between job submissions is small, it may not worth immediately switch-off nodes because the cost of switching can surpass the cost of just let it idle. Therefore, for better applicability, it must consider both the current workload on the system and the platform setup [14]. The same can be stated about the scheduling policy. There are policies best suited for specific usage patterns which also depends on the workload being executed. However, tailoring system-specific policies is not a straightforward task due to the fact of the workloads can suddenly change within days [11].

In order to address these issues, we propose a novel energy-aware scheduling policy that uses DRL to learn a system-specific policy. DRL methods learn a policy by interacting with a simulated environment through a trial-and-error procedure. The agent starts with no knowledge, each decision-making results on a stimulus (called reward) transmitting the quality of its actions on each of the observed states. Based on this stimulus, the agent tries to maximize its expected future reward by reinforcing the tendency to repeat the actions that led it to the most rewarded states [20].

Our DRL approach guides the jobs scheduler to optimize both perspectives, energy, and performance. Thus, we build an environment to simulate the Resource and Job Management System (RJMS) behavior. The environment is fed with real workload traces collected from Grid'5000 testbed [1]. In order to verify the generalization of our approach, we divided the traces into training and testing sets. We performed several experiments on both sets, the results were compared to rule-based scheduling policies which are commonly deployed on grid computing platforms.

This paper is organized as follows. Section 2 presents related work, Section 3 explains our formal notation. Section 4 details the problem formulation, and the learning method used. Section 5 describes the evaluation methodology. Section 6 presents the results, and discusses the policy learned by our method.

2 Related Work

The specialized literature covers trained Reinforcement Learning (RL) agents to detect subtle changes in the workload, adapting the management platform [9, 13]. Moreover, commonly used RL methods (*e.g.,* Q-learning, and SARSA) share a firm theoretical background. However, in complex problems using a large state space, simple RL mechanism are inefficient to learn an optimal policy without manual work on the reduction of the problem scope [16]. Given this limitation, some works apply DRL methods combining RL and Deep Learning (DL) to approximate a solution. This is the case of the DeepRM [12], which uses the REINFORCE method and a Deep Neural Network (DNN) to train an agent capable of scheduling jobs in order to minimize the slowdown. The main differences to our work are we evaluate our method using real workload traces and platform configuration, while DeepRM employed only a limited synthetic workload and platform.

DRL and DNN are also employed to train a workflow scheduler [10]. Instead of using a simple DNN, [10] implemented a pointer network to perform scheduling decisions, using information about the failure probability of each task, similar to our confidence level (Subsection 4.1).

In turn, DRL-Cloud uses Deep Q-Networks (DQNs) to train an agent to deal with the resource provisioning and task scheduling [4]. The main objective of DRL-Cloud is to minimize the energy cost in cloud computing providers, drastically differing from our workload and problem formulation. In grid computing a node can be individually provisioned for a user, while in cloud computing a single node can instantiate multiple Virtual Machines (VMs) from distinct users. Although there are plenty of different DRL-based schedulers, we identified a lack of studies specific for grid computing. In addition, most of the reviewed works are environment-specific, missing experiments using real workload and platform configurations. Thus, we propose a novel DRL-Based scheduler suited for grid computing and evaluate it with traces collected from a real grid platform.

3 System Model

Our experiments are based on the models from SimGrid [3] (grid system simulator), and Batsim [6], (RJMS simulator which uses SimGrid in the background).

3.1 Platform Model

Grid computing platforms can be described as a set of resources clustered in multiple geographically distributed sites. Users request a site, a cluster, a server, a processor or just a single core. Due to the fact of a core being the minimum a user can request, we characterized these resources as: *(i)* the computing capacity cpu_r, expressed in flop/s; *(ii)* the current state s_r; and *(iii)* the current power consumption, expressed in watts. In order to allow a coarse-grained control of resources, we

defined servers as independent clusters which own an unique set of resources with the same computing capacity and power profile. Servers can only be switched-off if (and only if) all of its resources are idle and only idle resources can start computing new jobs. Therefore, if a server is off it must be first initialized before attend to new users requests. A server is idle only when all of its resources are idle in the same way that it is computing if at least one of its resources is computing. The interconnection network was simplified, it is not take into account and the data transfers among resources of different servers occurs instantaneously. Finally, we follow the architecture design of Grid'5000 testbed and the platform is composed only by homogeneous resources. Figure 1 illustrates an example of the resource model and the transitions between states.

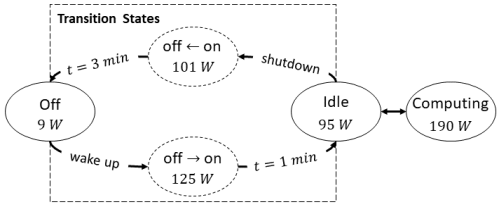

Fig. 1 Transitions between resource states and relative power consumption.

The power profile adopted (Figure 1) is based on experiments conducted on the Taurus cluster of Grid'5000 [17, p. 68] and are extensively used throughout this work to support the examples and experimental analysis. Following the Figure 1, a resource r can be in one (and only one) of the following states in a given time t, $s_r(t) \in S = \{computing, idle, off, on \rightarrow off, off \rightarrow on\}$. Each state has an associated power profile denoted by P_s and is expressed in watts. From the Figure 1, when the resource is computing it consumes $P_{computing} = 190$W, and while it remains idle it consumes $P_{idle} = 95$W on average.

Resources cannot be shared, meaning each job uses 100% of its allocated computing resources capacity. Idle resources can be switched off, which takes time $t_{on \rightarrow off}$ and consumes $P_{on \rightarrow off}$. Powered off resources must be switched on to start handling new jobs, which also takes time $t_{off \rightarrow on}$ and consumes $P_{off \rightarrow on}$. Thus, the energy consumption of a resource r depends only on its current state at time t and is given by $E_r(t) = \int P_s^r(t)dt$, expressed in joules. In this sense, the total energy consumption of the platform G is given by $E_G(t) = \sum_{r \in R} E_r(t)$.

3.2 Workload Model

In this work, the workload is composed of parallel and rigid jobs (J) online submitted and executed in batch mode. For each job $j \in J$, we consider the following characteristics: *(i)* the arrival time r_j (known at submission time); *(ii)* the number of requested computing resources q_j; *(iii)* the expected processing time $wall_j$ informed by the user (also called walltime); and *(iv)* the actual processing time p_j (known when the job finishes). In order to fully reproduce the behavior of the jobs in the traces, we defined the required amount of computation for a job j, expressed in flop/s, is a function of its processing time p_j and of the computing capacity cpu_r

of the allocated resources. Formally, this is given by $cpu_j = p_j * cpu_r$. Parallel jobs, which require more than one resource, the same amount of cpu_j is equally accounted for each allocated resource, resulting in the same given p_j. The RJMS does not know this information until the job finishes, when the job normally ends or when the walltime is reached. In either way, a job cannot be preempted, and the provisioned resources are only released when it is finished. In this sense, the $wall_j$ can be interpreted as an upper bound limit on the job j execution time.

4 DRL-based Scheduling

4.1 Problem Formulation

The learning phase occurs through the interaction of the agent with an environment, defined as a Markov Decision Process (MDP). A MDP is a mathematical framework for decision-making tasks which defines a tuple $M = (S, A, T, R)$, in which S is a finite state space, $A \neq \emptyset$ is a finite set of actions, $T : S \times A \to [0, 1]$ is a transition function and $R : S \times A \times S \to \mathbb{R}$ is an action-dependent reward function [20]. Following this framework, we assume the transition function is unknown and formulate the grid scheduling as follows:

State Space: The state $s(t)$ at time t is the union of the current queue state s_q, the current RJMS scheduling agenda s_a, and of the n past observations of the system state $s^p(t) = [s^p_{t-n}, ..., s^p_{t-1}, s^p_t]$. The system state includes information about: the number of current reserved resources; the total number of jobs in the queue; the current simulation time; and an indicative of the number of resources s_k in each state, *i.e.*, $s_k = [|r_{off}|, ..., |r_{computing}|]$. In this way, the agent can observe the elapsed time to a resource switch on-off and how much the queue is increasing within the time window (defined by n). Furthermore, the queue state includes information on: the number of resources requested q_j; the expected processing time $wall_j$ given by the user; and a confidence level which is the mean of the ratio between the actual processing time and the expected processing time of u past job submissions. Basically, the confidence level indicates the probability of the job j is going to execute less than the expected processing time. Lastly, the RJMS agenda includes information about jobs being executed, which are: the number of allocated resources; the remaining time based on the given expected processing time; and an estimation of the remaining time based on the confidence level.

Action Space: At each time step, the scheduler can select any of J jobs in the queue. Therefore, the action space is given by $A = \{a | a \in \{\theta, 1, ..., |J|\}\}$ in which $a = 1$ indicates the agent must schedule the first job in the queue, and $a = \theta$ means no job must be scheduled. In order to allow multiple scheduling decisions at a single time step, we freeze the simulation time when the agent is scheduling the jobs. The simulation proceeds only when $a = \theta$ or when the agent takes an invalid action, which occurs when a job cannot be scheduled because there is no resources available. Thus, we simplify the training phase by reducing the action space.

Reward Function: The reward function was designed to minimize the number of idle nodes while not degrading the overall performance. We formulate the reward function as a linearly-weighted combination of three metrics (Equation 1):

$$R = -\rho \times |N_{idle}| + \sigma \times \sum_{j \in Q} \frac{-1}{wall_j} + \tau \times \mu \tag{1}$$

This reward penalizes the agent proportionally to the number of current idle nodes ($|N_{idle}|$), and the number of jobs in the queue Q was normalized by its expected processing time $wall_j$. The mechanism saves energy by minimizing the set of active servers and the average slowdown (prioritizes small jobs). Furthermore, to encourage the agent to schedule the jobs, we give a bonus proportional to the utilization rate (μ,) which is the number of current computing resources. Parameters (ρ, σ, and τ) are adjustable weights for each of the mentioned perspectives.

4.2 Learning Method

The Proximal Policy Optimization (PPO) method [19] was adopted to train the agent through an actor-critic approach, in which N parallel actors are responsible for the decision-making, and a critic estimates the value of each observed state to evaluate actors' decisions. The policy and the value function are represented by artificial neural networks, differing in that the actors' network outputs a probability distribution over all possible actions, while the critic network applies a linear function.

The core idea is to increase the probability of selecting the actions which are better than the expected return estimated by the critic. The algorithm works as follows: *(i)* the actors collect experiences by individually running a policy $\pi_{\theta_{old}}$ for T timesteps while the critic estimates the value of each observed state $[V(s_1), ...V(s_T)]$; *(ii)* given the experiences and the critic estimations, an advantage function $\hat{A}_t$ tells the agent how much better its decision-making was facing to what is actually known; *(iii)* based on this signal a surrogate loss $L(\theta)$ is computed and optimized according to θ_{old} for K epochs. Then, the actors use the optimized policy π_θ to collect new experiences and the whole procedure is repeated.

The surrogate loss used by the PPO method is defined in Equation (2). This function modifies the first term by clipping $r_t(\theta)$ at $1-\varepsilon$ or $1+\varepsilon$, based on the $\hat{A}_t$ estimation. Formally defined in Equation (3), $r_t(\theta)$ gives the ratio between the policy with the actual parameter vector θ and the old parameter vector θ_{old}, used to collect the experiences.

$$L(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t] \tag{2}$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{3}$$

Consequently, changes which would make the objective improve beyond the clipped value are simply ignored, and the same experiences collected by a policy $\pi_{\theta_{old}}$ can be used to perform multiple steps of optimization without completely destroying the policy due to large updates [19].

5 Evaluation Methodology

5.1 Metrics

The performance of each scheduling policy was evaluated based on three metrics. The first one is the total amount of energy spent by idle and switching nodes. Idle and switching nodes consume non-negligible amounts of energy while they are neither serving providers nor users' needs. Furthermore, the waste of energy increases proportionally to the time nodes were idle and to the number of shutdown events. Thus, these metrics presents how much energy was completely wasted, and can be used as an indicator of the policy energy-efficiency.

The second metric is the waiting time, which measures the time a job j waited in the queue, as given by Equation (4). This is a job-oriented metric and its average serve as an indicator of the performance of the system. Specifically, $start_j$ is the time the job j was scheduled to start. A problem with the waiting time is it does not present any distinction between jobs with different execution times. Therefore, it is not useful as an indicator of scheduler fairness. Keeping this is mind, a commonly used metric is the slowdown, which measures the ratio between the time that a job spent on the system $(wait_j + p_j)$ and its actual processing time (p_j). However, the slowdown does not take into account the number of resources requested by each job. Thus, for the parallel batch scheduling problem the per-processor slowdown $(pp-sld)$ is a more appropriate metric [2], given by Equation (5).

$$wait_j = start_j - r_j \tag{4}$$

$$pp-sld_j = \max(\frac{wait_j + p_j}{q_j \times p_j}, 1) \tag{5}$$

The interpretation of the pp-slowdown is the waiting time of a job j must be proportional to the time it spent executing on the allocated resources. Therefore, by considering these three metrics we can evaluate the energy-efficiency, performance, and fairness perspectives.

5.2 Simulation Setup

We simulate the behavior of a RJMS using an extended-version of Batsim, called GridGym[1]. The extension added the OpenAi Gym framework providing grid-specific environments suited for RL tasks. We followed the models described in Section 3 to setup GridGym and implemented a timeout policy to switch-off servers after 5 minutes of idling time, which corresponds to the inter-arrival time we characterized a sequential job. Thus,it is possible to identity the impact of executing immediate scheduling decisions on energy consumption.

Real workload traces were used to evaluate the policies, the platform configuration follows the original hardware settings[2], as per-cluster summarized on Table 1. The traces were organized in a daily-basis, and only days which had at least two job

[1] Information available on https://github.com/lccasagrande/GridGym

[2] Information available on https://www.grid5000.fr/w/Hardware

submissions were considered. For each trace, it was removed the jobs which did not execute, requested a larger amount of resources than the ones offered by the cluster, or set an invalid walltime. Moreover, the traces were grouped (on each cluster) into five categories as function of the occurrences of sequential job submissions on that day. The last five columns of Table 1 shows the number of days for each of these groups. Formally, sequential job submissions are separated by a small period of time (less than 5 minutes). Such jobs can increase the energy consumption of a platform by forcing the RJMS to turn on nodes for hosting them. Therefore, by splitting the traces we can mitigate the eventual over-fitting on specific workloads.

Table 1 Real workload traces from GRID'5000 used for evaluation.

Cluster	Period	# Cores	# Jobs	Seq. Jobs %	Util. %	Group				
						#1	#2	#3	#4	#5
Orion	10/12 - 10/19	48	66,005	39.7 %	62.7 %	659	777	496	129	28
Graphite	12/13 - 10/19	64	60,786	56.0 %	47.3 %	487	705	427	149	24
Econome	04/14 - 10/19	352	66,557	66.8 %	57.6 %	259	520	504	271	39

In order to evaluate the performance, we compared our proposal to three commonly used scheduling policies: *(i)* First-Fit schedules the first job ready to execute; *(ii)* EASY schedules jobs in a FIFO order and use a backfilling mechanism to handle jobs in the queue which can immediately start without delaying the next first job; and *(iii)* a Smallest Estimated Area First (SAF) policy as a variation of the EASY sorting the backfilling queue based on the estimated area of each job j given by $wall_j * q_j$. Each scheduling policy has the same view of the system. We scaled up the time in the traces to minutes to accelerate the training and evaluation process. Therefore, the simulation time proceeds in one minute after the scheduler finishes its procedure. This process occurs only when there are jobs in the queue. Otherwise, the simulation goes to next event time, *i.e.,* a new job submission / shutdown event.

5.3 DRL Parameters

During the training phase, were instantiates 16 parallel actors to collect experiences. The actor and critic networks are independent but follow the same architecture. This architecture is similar to a sequence-to-sequence (seq2seq) model, composed of an encoder and a decoder. In order to simplify and speed up learning, we do not implement the decoder and the hidden state vector of the encoder is passed through a dense layer in a sequence-to-element architecture. Thus, the network configuration consists of three parallel Gated Recurrent Unit (GRU) layers with 64 units each that are followed by a dense layer with 128 units. Each of the GRU layers is responsible for handling a specific input of the observation given by the environment, which are: a sequence of the first 20 jobs in the queue; a sequence of the 20 earliest jobs to be finished, given by the RJMS agenda; and the last 20 snapshots of the system state. This sequence size is not limited to our single choice and could be increased. Through simple tests, we found these values suitable to demonstrate the potential of DRL-Based schedulers. Moreover, information about the time required to switch nodes On-Off is also encompassed into the last 20 snapshots of the system.

Regarding the PPO method parameters, we used the Generalized Advantage Estimator (GAE) [18] as the advantage function with the discount factor γ set to 0.97 and the λ set to 0.95. The ε parameter of the surrogate loss (Equation 2) is 0.20. The algorithm is trained for 25 million timesteps and the optimization is done every 360 timesteps for 6 epochs with a batch size of 180. Regarding the weights of the reward function, we set both ρ and σ to 1 while the τ is set to 0.2 which, is in accordance with the idling time defined on the timeout policy (Section 5.2). Consequently, scheduling decisions initiating an off server will be compensated if the job executes for at least the defined idling time on the timeout policy.

6 Results & Discussion

Figure 2 plots the cumulative daily average energy waste, waiting time, and pp-slowdown for each cluster trace. The results for DeepScheduler are the mean of ten consecutive experiments, and the middle solid lines represent the cumulative average of the respective metric, while the vertical dashed lines separate the training and testing results. Thus, to minimize the effect of outliers on the job-centric metrics (waiting time and $pp-slowdown$), we present only the scheduling results within the 10-90 percentile range. Finally, the cumulative maximum and minimum values are represented by the upper and lower horizontal dashed lines.

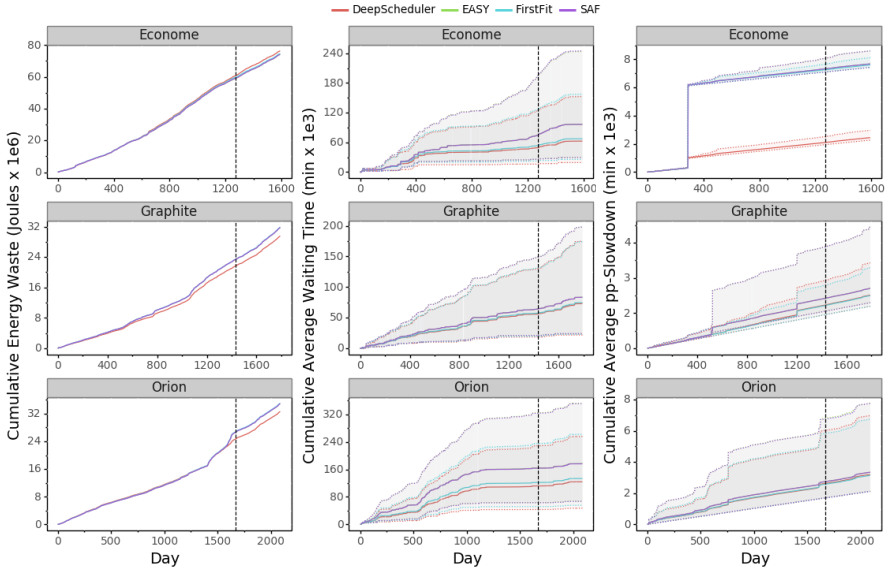

Fig. 2 Cumulative daily average energy waste, waiting time and pp-slowdown for each trace.

DeepScheduler was able to achieve better energy savings on most of the traces when compared to the other policies (Figure 2). In Orion and Graphite experiments, DeepScheduler achieved energy savings up to 6.5–7.0% respectively, while on the Econome trace DeepScheduler spent 2.9% more energy than EASY and SAF. As expected, the energy efficiency of the SAF, EASY and First-Fit scheduling policies are quite the same. These policies are not energy-aware, therefore we do not expect it to be much different from each other.

Comparing the average waiting time, the DeepScheduler was acting more likely to a First-Fit policy. Both SAF and EASY scheduling policies had poor results and the First-Fit seems the best scheduling policy to apply on the selected traces. The DeepScheduler was able to learn and provide better results than First-Fit on some days. On average, the DeepScheduler minimized the average waiting time by 5.8% in comparison to the First-Fit and by 27% in comparison to the SAF and EASY policies. One point that is worth noticing is how much energy can be saved while keeping better or equal performance if a system-specific policy is chosen. On Graphite and Orion traces, the DeepScheduler was able to both save energy and improve the performance of the system.

Regarding the fairness metric, it was noted a subtle performance downgrade on the results using the SAF and EASY scheduling on the Econome traces. The main reason behind this is both (SAF and EASY) scheduled the jobs in a First Come First Served (FCFS) order. Therefore, if the first job in the queue has requested all the resources available for a long time (*i.e.*, 20 hours) the slowdown of the remaining jobs will inflate the metrics. Thus, to overcome this situation, one must change the ordering policy and implement another mechanism to prevent starvation, let as future work. Furthermore, in the Econome traces, there were only 9 days that the load surpassed 300% of the platform capacity which, represents less than 1% of the workloads, and were considered outliers. However, we decided to include these traces to check if the DeepScheduler could surpass this limitation without any manual work. Indeed, its performance on the Econome trace was better while on the other traces it presents similar results to the First-Fit policy.

An important question that raises with DRL-Based scheduling policies concerns the behavior learned by the agent. To give an insight into this point, Figure 3 presents the slowdown distribution for all jobs of each scheduling policy for each trace.

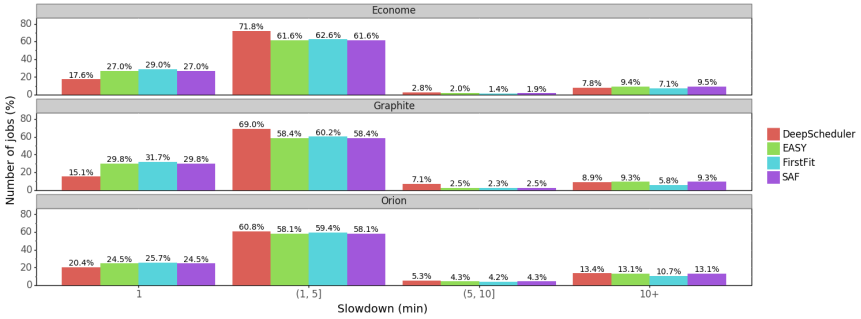

Fig. 3 Distribution of the slowdown values for each trace.

The jobs were separated into buckets in function of its slowdown (Figure 3). DeepScheduler exhibits a high concentration of jobs with slowdown values whiting the 1–5 range while there is a low concentration of jobs with a slowdown of 1. A characteristic of the used traces is the high percentage of sequential jobs which were submitted within 5 minutes and mostly finished prematurely. Thus, it makes sense to delay some jobs in the expectation that a job running will release its resources prematurely in order to minimize the number of active resources. Following this

principle, the DeepScheduler delays more jobs than the other policies in order to optimize its energy savings.

Delaying some jobs in order to save energy is interesting but there is no guarantee of a job will finish its execution prematurely. Moreover, this strategy must be taken with caution to not considerably degrade the system performance for nothing. Figure 4 presents the plot of the slowdown values for all sequential jobs of each scheduling policy, it clarifies how the DeepScheduler is deciding to delay the jobs. In order to diminish the effect of outliers, the InterQuartile Range (IQR) rule was applied and only slowdown values within 1.5 x IQR were considered.

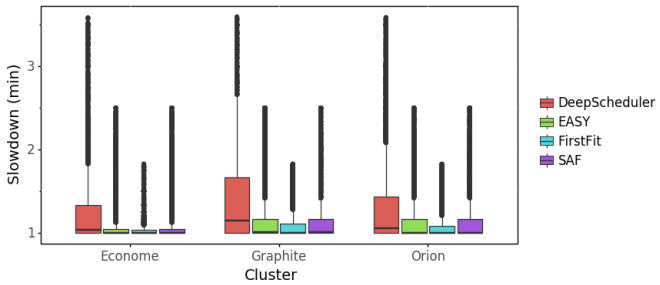

Fig. 4 Slowdown values for all sequential jobs.

Figure 4 allows to observe DeepScheduler exhibits the highest slowdown values for the sequential jobs on all traces while the First-Fit policy gives the lower values. Therefore, we can argue that the DeepScheduler is waiting before taking a scheduling decision and the sequential jobs are the most prone to be delayed. By not taking immediate decisions, the scheduler can explore other alternatives that may occur when a job finishes prematurely or a small job in length is submitted within the next minutes. DeepScheduler was able to learn how to behavior following this principle.

7 Considerations & Future work

In this paper, we explored a DRL method as an alternative to build an adaptive energy-aware scheduling policy. By exploiting the workload patterns and tailoring a system-specific policy, better energy savings were achieved in comparison to commonly used rule-based policies. Our results indicate a DRL-based scheduling policy is a feasible choice when compared to traditional schedulers. As future work, we will address specific workloads based on a variant degree of sequential jobs.

Acknowledgements This study was supported by FAPESC, UDESC, and LabP2D. Experiments were carried out on the GRID'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

References

1. Bolze, R., Cappello, F., Caron, E., Dayd, M., Desprez, F., Jeannot, E., Jgou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.G., Touche, I.: Grid'5000: A large scale and highly reconfigurable experimental grid testbed. The International Journal of High Performance Computing Applications **20**(4), 481–494 (2006)

2. Carastan-Santos, D., Yokoingawa De Camargo, R., Trystram, D., Zrigui, S.: One can only gain by replacing easy backfilling: A simple scheduling policies case study. In: Cluster, Cloud and Grid Computing (CCGrid), 2019 19th IEEE/ACM International Symposium on (2019)
3. Casanova, H., Giersch, A., Legrand, A., Quinson, M., Suter, F.: Versatile, scalable, and accurate simulation of distributed applications and platforms. Journal of Parallel and Distributed Computing **74**(10), 2899–2917 (2014)
4. Cheng, M., Li, J., Nazarian, S.: Drl-cloud: deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In: Proceedings of the 23rd Asia and South Pacific Design Automation Conference, pp. 129–134. IEEE Press (2018)
5. Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys & Tutorials **18**(1), 732–794 (2016)
6. Dutot, P.F., Mercier, M., Poquet, M., Richard, O.: Batsim: a realistic language-independent resources and jobs management systems simulator. In: Job Scheduling Strategies for Parallel Processing, pp. 178–197. Springer (2015)
7. Galizia, A., Quarati, A.: Job allocation strategies for energy-aware and efficient grid infrastructures. Journal of Systems and Software **85**(7), 1588 – 1606 (2012). Software Ecosystems
8. Hinz, M., Koslovski, G., Miers, C., Pilla, L., Pillon, M.: A cost model for iaas clouds based on virtual machine energy consumption. Journal of Grid Computing **16**(3), 493–512 (2018)
9. Hussin, M., Hamid, N.A.W.A., Kasmiran, K.A.: Improving reliability in resource management through adaptive reinforcement learning for distributed systems. Journal of parallel and distributed computing **75**, 93–100 (2015)
10. Kintsakis, A.M., Psomopoulos, F.E., Mitkas, P.A.: Reinforcement learning based scheduling in a workflow management system. Engineering Applications of Artificial Intelligence **81**, 94 – 106 (2019)
11. Legrand, A., Trustram, D., Zrigui, S.: Adapting batch scheduling to workload characteristics: What can we expect from online learning? In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 686–695 (2019)
12. Mao, H., Alizadeh, M., Menache, I., Kandula, S.: Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50–56. ACM (2016)
13. Moghadam, M.H., Babamir, S.M.: Makespan reduction for dynamic workloads in cluster-based data grids using reinforcement-learning based scheduling. Journal of computational science **24**, 402–412 (2018)
14. Orgerie, A., Lefvre, L., Gelas, J.: Save watts in your grid: Green strategies for energy-aware framework in large scale distributed systems. In: 2008 14th IEEE International Conference on Parallel and Distributed Systems, pp. 171–178 (2008). DOI 10.1109/ICPADS.2008.97
15. Orgerie, A.C., Assuncao, M.D.d., Lefevre, L.: A survey on techniques for improving the energy efficiency of large-scale distributed systems. ACM Computing Surveys (CSUR) **46**(4), 47 (2014)
16. Orhean, A.I., Pop, F., Raicu, I.: New scheduling approach using reinforcement learning for heterogeneous distributed systems. Journal of Parallel and Distributed Computing **117**, 292–302 (2018)
17. Poquet, M.: Simulation approach for resource management. Theses, Université Grenoble Alpes (2017). URL https://tel.archives-ouvertes.fr/tel-01757245
18. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)
19. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
20. Sutton, R.S., Barto, A.G., Williams, R.J.: Reinforcement learning is direct adaptive optimal control. IEEE Control Systems **12**(2), 19–22 (1992)
21. Vicat-Blanc Primet, P., Anhalt, F., Koslovski, G.: Exploring the virtual infrastructure service concept in Grid'5000. In: 20th ITC Specialist Seminar on Network Virtualization. Hoi An, Vietnam (2009)