Don't Hurry Be Green: Scheduling Servers Shutdown in Grid Computing with Deep Reinforcement Learning

Bob

Graduate Program
E-mail: bob@domain.com

Abstract: Grid computing platforms dissipate massive amounts of energy. Energy efficiency, therefore, is an essential requirement that directly affects its sustainability. Resource management systems deploy rule-based approaches to mitigate this cost. However, these strategies do not consider the patterns of the workloads being executed. In this context, we demonstrate how a solution based on Deep Reinforcement Learning is used to formulate an adaptive power-efficient policy. Specifically, we implement an Off-Reservation approach to overcome the disadvantages of an aggressive shutdown policy and minimize the frequency of shutdown events. Through simulation, we train the algorithm and evaluate it against commonly used shutdown policies using real traces from GRID'5000. Based on the experiments, we observed a reduction of 46% on the averaged energy waste with an equivalent frequency of shutdown events compared to a soft shutdown policy.

Keywords: Deep Reinforcement Learning; Grid Computing; Energy-Aware Scheduling; Shutdown Strategy; Markov Decision Process; Resource Management.

1 Introduction

Grid computing provides massive on-demand computing power by sharing resources among multiple geographically distributed institutions. Virtual organization groups allow sharing, discovery, and allocation of computing resources among providers and consumers. Not surprisingly, grids became a popular solution to handle intensive parallel applications and large-scale scientific experiments [18, 50].

Sharing resources to constitute large-scale platforms is an appealing and affordable choice to increase the number of resources available to a given application. By adding more providers, a grid platform can easily range from a few hundred (e.g., DAS-4 [1]) to thousands of CPU cores (e.g., GRID'5000 [4]). Such paradigm allows users to scale up their applications to solve more complex problems, but at a price [19]. The energy consumption increases almost linearly with resource utilization. Taking into consideration the size of the platform, such consumption can easily become unfeasible if not tackled [46]. For example, operating a thousand processors fluctuating from 95 W at idle to more than 200 W under stress leads to a power consumption of 95 kW to 200 kW [39, p. 68]. Even not considering the environmental impact and the costs with cooling, there is a non-negligible financial cost associated to just keeping such platform powered on. In this way, power bills have become a significant expense that affects the sustainability of the infrastructure [11]. Therefore, energy-efficient strategies are one of the main interests of system administrators and providers [22, 48].

Achieving better energy efficiency not only depends on the choice of efficient hardware but also on the management strategies deployed on different levels of the platform [32, 34]. As an example, the energy consumed by underutilized servers (called nodes) can be minimized by deploying Dynamic Voltage Frequency Scaling (DVFS) techniques at the job level. Such technique can scale down the processor frequency based on its usage pattern to allow energy savings when jobs are not executing computation intensive tasks [23, 52]. At the scheduler level, power capping or energy budget techniques can be used to keep the energy consumption under a threshold by limiting the usage of the platform [5]. In addition, energy-aware job scheduling policies can be adopted to improve the energy efficiency using multi-constraint objectives [45]. At platform level, shutdown techniques use the time between jobs to minimize the number of idle nodes. Idle nodes consume a considerable amount of energy, therefore simply turning them off leads to potential energy savings [3, 21, 40, 49].

Energy minimization has been vastly studied through the years and there are plenty of different Dynamic Power Management (DPM) strategies available. Among the most popular, shutdown is one of the most promising solutions due to its expected higher impact on energy efficiency [2]. Shutdown strategies turn off nodes that are idle for a time, therefore consuming considerably less energy than powered on but idle nodes. Moreover, there are no gains on keeping nodes powered on if they are not going to be used for a long time. Keeping these nodes on only wastes energy while neither serving providers nor users needs. Such assumption would enforce the adoption of an aggressive shutdown policy, turning off nodes immediately after they become idle. However, in some situations aggressive policies are not the best solution [33].

Identifying the moments to turn off nodes depends on the workload and on the infrastructure of the platform. For example, if a node takes 5 minutes to shutdown, 10 minutes to wake up, and the job arrival time is less than or equal to 15 minutes, there is no benefit in using an aggressive shutdown policy because the node keeps switching between states most of the time. Moreover, the energy consumed while switching can be higher than if it just stayed idle. To overcome this situation, one must consider the peculiarities of each system [27, 41]. However, using historical data is not a straightforward task due to the dynamicity of grid systems and workloads. Consequently, the solutions typically involve using simple but scalable rule-based policies that are not based on workload patterns. Applying the same set of rules on systems with different infrastructures and usage patterns leads to an underoptimized setup [26]. Adaptive strategies address this issue [25, 36].

Regarding adaptive control, Reinforcement Learning (RL) introduces methods in which an agent can learn a policy through a trial and error procedure. The agent starts with no knowledge. At each interaction with the environment the agent receives a stimulus (called reward) based on the quality of its decision. This is based on the common sense notion that if one chooses an action that is followed by a satisfactory state, then the tendency to execute that same action should be reinforced. As the number of interactions goes to infinity, the agent tries to learn a policy that maximizes its expected (sometimes discounted) future reward [47]. This capacity makes RL a possible solution to dynamically adapt the policy to distinct scenarios, but it has some limitations that may prevent it from learning optimal policies on problems with large state spaces. Due to the complex nature of a grid system, such strategy may not fit without manual work on the reduction of the problem scope [37]. In such cases, Deep Reinforcement Learning (DRL) techniques that joins RL with Deep Learning (DL) are able to approximate a solution [30].

In this context, we propose a novel server shutdown strategy based on DRL named **DeepShutdown**. DeepShutdown operates at the platform level along with the scheduling policy to determine when nodes must be turned off and for how long they must be kept in that state. We use an Off-Reservation (OR) approach that works similar to a load consolidation algorithm, but that reserves a subset of nodes to itself. Nodes reserved cannot be used by the job scheduler, which will consequently concentrate the queue load on the remaining nodes. In this way, it's possible to save energy while minimizing the number of On-Off cycles by adapting the number of available nodes. DeepShutdown exploits the workload to learn the moments when it worth to increase/decrease the nodes available for scheduling the grid jobs in order to save energy by concentrating the load on fewer nodes. We relax the assumption that a job must execute as soon as possible and give the algorithm a upper bound limit on the waiting time for each job. DeepShutdown is allowed to delay some jobs up some point if the energy economy pays off.

By considering the peculiarities of each workload and allowing the extra delay in the start of some jobs, we achieve considerable energy savings. Moreover, although some arguments discourages prediction methods for servers shutdown [40] we demonstrate how it can be successfully employed to achieve better results than pure shutdown policies. In order to validate this, we conduct several simulations using real workload traces from the GRID'5000 testbed [4] and we estimate the impact of the workload on different shutdown techniques. From the results, we observe an averaged increase of 11.7% on energy savings with 17.9% less shutdown events in comparison to an aggressive shutdown policy. By comparing it with a soft policy, the savings achieved by DeepShutdown increases up to 46% on average while the number of shutdown events increases only 4%.

In short, the main contributions of our work are:

- We address the influence of job submission pattern and job failures on a rule-based shutdown policy by analyzing the GRID'5000 traces.
- We demonstrate how an OR technique can explore workload properties to improve shutdown-based policies.
- We demonstrate how the shutdown problem can be modeled into a Markov Decision Process (MDP) and used to learn a policy through state-of-the-art DRL techniques.
- We present a shutdown strategy that can adapt to the usage pattern of the platform and we conduct several experiments to evaluate this adaptability.
- We show how DRL can be used to optimize the grid resource and jobs management.
- We extend Batsim [13] to deal with the peculiarities of DRL techniques by joining it with OpenAi Gym [6]. This integration (named GridGym) provides a series of ready-to-use environments that deal with problems commonly faced by resource management systems.

The rest of this paper is organized as follows. Section 2 details the motivation and describes the GRID'5000 traces. Section 3 explains the models and formal notation. Section 4 details DeepShutdown and the MDP formulation. Section 5 describes the simulation environment and presents the results. Section 6 discusses the policy learned by the DeepShutdown. Section 7 presents related work. Finally, Section 8 concludes this work and presents future directions.

2 Motivation and Problem Definition

This section presents the motivation behind our work. We highlight the disadvantages of pure shutdown policies and we explore alternatives that could be considered. We support our claims by analyzing a set of GRID'5000 traces.

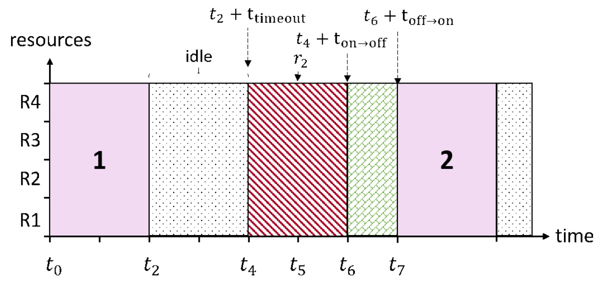
2.1 Disadvantages of Pure Shutdown Policies

The power consumption of idle resources has a significant impact in the overall energy consumed by a computing platform. To overcome this situation, a common choice is to turn off resources after an idle period of time. Although this timeout strategy (also called *opportunistic shutdown* [12] reduces the energy wasted on idle periods, it can also degrades performance and increases the energy consumption in some situations [33].

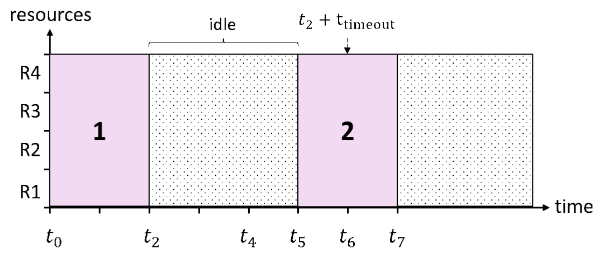
In order to illustrate this, Figure 1 demonstrates an example with two timeout policies that differ only on the idle time ($t_{timeout}$) that must pass before a resource is turned off. Assume that $t_0, t_1, ... \in T$ are discrete time representations, r_j is the submit time of job j, $R_1, R_2, ... \in R$ are computing resources, $t_{on \rightarrow off}$ is the time to completely switch off a resource, $t_{off \rightarrow on}$ is the boot up time, P_s is the power consumption at state s and $P_{off} < P_{idle} < P_{on \rightarrow off} < P_{off \rightarrow on} < P_{computing}$.

Starting with Figure 1a, a timeout policy with $t_{timeout} = 2t$ is deployed. At t_0 all resources are being used by job 1 and the job queue is empty. At t_2 job 1 finishes and the resources remain idle until t_4. The idling time ($2t$) is observed and the timeout policy switches them off. The resources take from t_4 to t_6 to switch off and at t_5 job 2 is submitted but it cannot start because there is no available resource at the moment. At t_6 the state transition is completed and the resources are switched on to compute job 2. The resources take from t_6 to t_7 to switch on and job 2 can finally start. In this small example, the timeout policy increased both job 2 waiting time and the amount of energy consumed.

In Figure 1b we illustrate a scenario in which this situation may be mitigated by simply increasing the idling time of the policy to $t_{timeout} = 4t$. From t_0 to t_2 all resources are being used by job 1. Than, the resources remain idle for $3t$ until job 2 is submitted at t_5. In this case, the resources are not switched off because the idling time is below the threshold. Job 2 can immediately start. The time resources spent in idle state compensate the cost of switching on and off and job 2 was not delayed. If no other job is submitted to the system, after the idle time surpasses the threshold all resources will be switched off, therefore saving energy. However, there is no guarantee that another job will not arrive right after. To avoid this situation, the idling time must be a function of the actual workload. More specifically, the solution must have information about future jobs to determine if it compensates to switch off resources [41]. Such information is complex to obtain



(a) Timeout policy with $t_{timeout} = 2t$



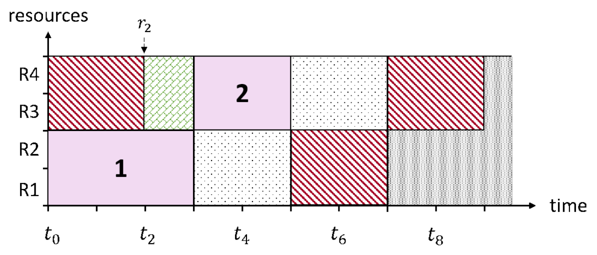
(b) Timeout policy with $t_{timeout} = 4t$

Figure 1: Illustration of the effectiveness of a pure timeout policy

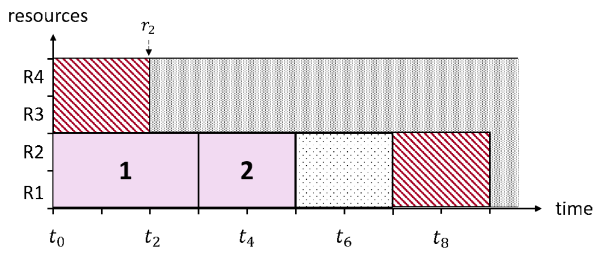
or predict since grid systems (application workload and resource utilization) dramatically change over time [26].

An alternative to overcome this situation is illustrated in Figure 2. In Figure 2a a timeout policy is deployed and the same undesired behavior can be seen. In this case, r_1 and r_2 are being used by job 1 while r_3 and r_4 are switching off. At t_2 job 2 requested two resources and r_3 and r_4 are switched on for it. The boot up time takes $1t$, so job 2 can start at t_3. When job 2 starts, job 1 unexpectedly release its allocated resources that remain idle until the idling time is observed by the timeout policy. The same applies to the resources allocated for job 2 after it finishes at t_5. If the scheduler could know the execution time of job 1 beforehand, the unnecessary switch on of resources r_3 and r_4 could be avoided by simply delaying job 2 execution by $1t$. Additionally, the energy wasted by these resources while they are idle or switching off can also be minimized.

It is possible to apply an OR approach instead of directly predicting the execution time of each job or the submission of future jobs. Figure 2b illustrates an OR approach that reserves some resources to itself in order to delay job 2 execution and avoid the unnecessary switch on of resources r_3 and r_4. An advantage of this method is that some jobs could be forced to wait until the switching on cost pays off, but the problem arises then in finding a balance between degrading performance and decreasing energy consumption. In this case, the algorithm must figure out how long it is worth delaying the execution of some jobs in order to save energy without dramatically increasing the jobs waiting time. Either way, the solution is able to achieve higher energy efficiency than pure timeout policies if the workloads contain a high number of sequential submissions. Performing such a task is not



(a) Timeout policy with $t_{timeout} = 2$



(b) Timeout policy with an OR approach

Figure 2: Illustration of the effectiveness of an off reservation approach

straightforward, but we demonstrate how this can be achieved with DRL in the next sections.

2.2 The Impact of Sequential Jobs Submissions

The potential energy savings achieved by an OR approach depend on the current workload in the system. As long as there are sequential jobs submissions, the strategy works. Formally, sequential job submissions are separated by a small period of time (less than 5 minutes in this work). Moreover, the jobs may belong to multiple users and request a distinct number of resources. We're more interested in the inter-arrival time.

Sequential jobs can increase the energy consumption of a platform by forcing the resource management system to turn on nodes for hosting them. If the job has a small execution time, the energy spent in switching on and the energy consumed while it stays idle after being released are completely wasted. In order to exemplify this situation we performed an analysis with the management and scheduling traces collected from GRID'5000[1] sites.

Table 1 summarizes some statistical properties of each trace. The traces correspond to the entire operation time, since its date of arrival[2]. For each trace we removed the jobs that did not execute, requested a larger amount of resources than the ones offered by the cluster, or set an invalid walltime (the upper bound time for the request). In order to analyze the number of sequential jobs we organized the traces per day. The last column of Table 1 shows the percentage of such jobs in each trace.

All clusters had an average utilization rate. Clusters Grisou and Taurus had a larger number of job submissions than others. All clusters had a considerably high rate of sequential jobs – only Hercule and Orion are below 50%. The overall mean was about 58.4%, which

Table 1 Traces Summary.

Site	Cluster	Period	# Cores	# Jobs	Seq. Jobs %	Util. %
Lyon	Taurus	Sep 12 - Oct 19	168	81,925	54.1 %	51.7 %
Lyon	Hercule	Oct 12 - Oct 19	48	50,003	46.9 %	49.8 %
Lyon	Nova	Feb 17 - Oct 19	368	42,171	52.9 %	58.3 %
Lyon	Orion	Oct 12 - Oct 19	48	66,005	39.7 %	62.7 %
Nancy	Graphite	Dec 13 - Oct 19	64	60,786	56.0 %	47.3 %
Nancy	Grimoire	Jan 16 - Oct 19	128	36,986	52.7 %	56.4 %
Nancy	Grisou	Jan 16 - Oct 19	816	99,291	75.4 %	69.5 %
Nantes	Econome	Apr 14 - Oct 19	352	66,557	66.8 %	57.6 %
Nantes	Ecotype	Jan 18 - Oct 19	960	39,868	81.1 %	66.5 %

indicates that the majority of the jobs are sequential submissions. In order to better analyze this behavior, we decided to select the oldest clusters of each site (Taurus, Orion, Graphite and Econome) which also give us a wide range of platform sizes.

In Figure 3 we show the occurrences of sequential jobs for each of the chosen clusters in a year basis along with the status it ended up. We removed the information from incomplete years to give a fair comparison. The occurrences of sequential jobs are not seasonal: every year there are more than 30% of sequential jobs and some years are almost completely dominated by them (like in Econome, 2015). We observe a high rate of jobs that ended up in an error state. The number of failures cannot be neglected and represent the majority of the jobs in the traces. Such behavior reinforces the idea of burst submissions, which can be characterized by sequential submissions of the same job by a single user. If the job ended with an error, the submissions are commonly repeated and the number of sequential jobs naturally increases. This behavior can be harmful in systems based on aggressive timeout policies. In such cases, if the node being used is turned off between the interval of the sequential submissions, more energy is wasted. Our work reduces energy waste by delaying the execution of sequential jobs if they do not pay off the cost of switching on the nodes.

3 Grid, Workload, and Energy Models

This section provides background information on systems, workloads, and energy models. Following the specialized literature, the description is based on the models from SimGrid [8] and Batsim [13].

3.1 Grid Platform Model

A grid platform is composed of a set of resources clustered in multiple geographically distributed sites. Each site contains one or more clusters with an arbitrary number of computing servers (called nodes). Each node is composed by a set of processors that contains one or more cores. Users can request a whole cluster, a node, a processor or just a single core. Therefore, cores are

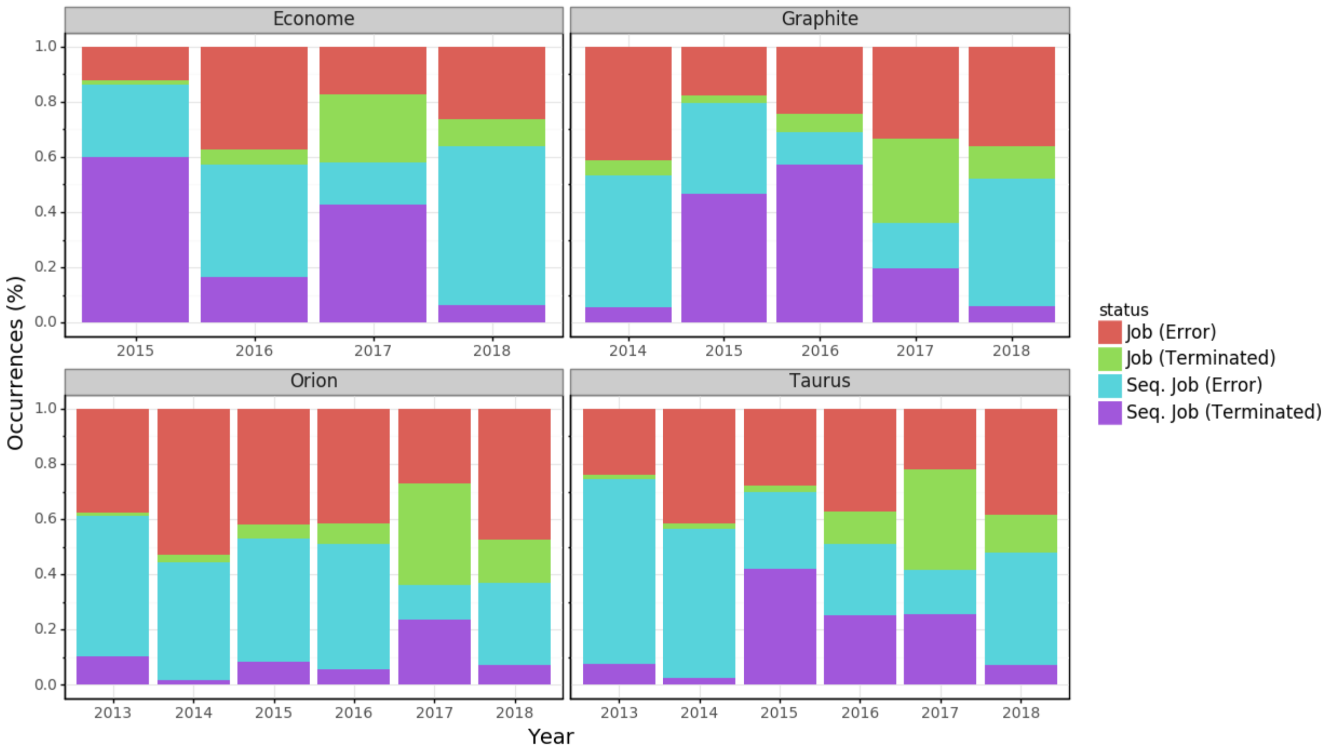


Figure 3: Occurrences of sequential jobs in each analyzed cluster trace per year.

simply referred to as resources r and are characterized by: *(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.

This work focuses on homogeneous clusters. The interconnection network is not considered. Therefore, each node has the same number of resources and each resource has the same computing capacity and power profile. Servers are independent and each one is composed of a unique set of resources. In this sense, servers can only be turned off if (and only if) all resources of the same server are in the idle state. When the node is initialized, all of its resources are also switched on. Therefore, a node is idle only when all of its resources are idle in the same way it is computing if at least one of its resources is computing.

Figure 4 illustrates the resource model and the transitions between states. A resource r can be in only one of the following states in a given time instant $s_r(t) \in S = \{computing, idle, off, on \rightarrow off, off \rightarrow on\}$. Each state s_r has an associated power profile denoted by P_s and expressed in watts. Using the GRID'5000 data as example, when the resource is computing it consumes $P_{computing} = 190$W and while it remains idle it consumes $P_{idle} = 95$W on average.

Only idle resources can start computing jobs. Resources cannot be shared, meaning that each job uses 100% of its allocated resources computing capacity. Idle resources can be switched off, which takes time $t_{on\rightarrow off}$ and consumes $P_{on\rightarrow off}$. In the example presented at Figure 4, the resource takes 3 minutes to completely

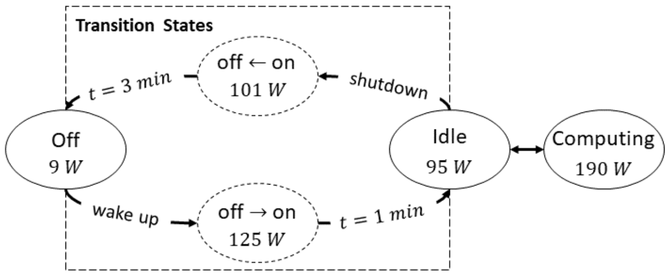


Figure 4: Transitions between resource states and relative power consumption.

shutdown and consumes 101W during the transition between states. Powered off resources must be switched on to start handling new jobs, which also takes time $t_{off\rightarrow on}$ and consumes $P_{off\rightarrow on}$. In the example, the resource takes 1 minute to boot up and consumes 125W. Resources transitioning between on and off become unavailable until they completely finish the transition. The power profile adopted in Figure 4 is based on experiments conducted on the Taurus cluster [39, p. 68] and are used throughout this work to support the examples and experimental analysis.

3.2 Workload Model

The workload is composed of a set J of parallel and rigid jobs in which are submitted online and execute in batch mode. For each job $j \in J$, we consider the following characteristics:

- The arrival time r_j of the job (only known when the job is submitted);
- The number of requested computing resources q_j;
- The expected processing time $wall_j$ informed by the user (also called walltime); and
- The actual processing time p_j (only known when the job finishes).

In order to fully reproduce the behavior of each job in the traces we consider that the total amount of computation done is a function of the processing time p_j and of the computation capacity cpu_r of the allocated resources. Formally, the amount of computation is given by $cpu_j = p_j * cpu_r$. For parallel jobs, which require more than one resource, the same amount of computation cpu_j is equally computed on each allocated resource resulting in the given p_j. In both cases, the Resource and Job Management System (RJMS) does not know this information until job finishes. A job cannot be preempted and the provisioned resources are only released when it finishes or when the $wall_j$ expire. In the last case, the resource manager forces the job to finish.

3.3 Energy Model

The total power consumption of a platform G at time t only depends on the state $s_r(t)$ of each resource $r \in G$. The energy consumption of a resource r is given by $E_r(t) = \int P_s(t)dt$, expressed in joules. For example, following the profile in Figure 4 the energy consumed by a single resource to switch off is $E_r = 303$ joules while the energy spent on boot up is equal to $E_r = 125$ joules. Therefore, the total energy consumption of the platform G is given by $E_G(t) = \sum_{r \in R} E_r(t)$. This model is a special case of the model adopted in SimGrid [8] in which resources can be idle (load= 0%) or at full load (load= 100%) when powered on.

4 DRL-based Power Management

This section details our proposed method. First we introduce the MDP formulation, followed by the explanation of how we implemented the algorithm within the RJMS. Lastly, we give some details about the training algorithm to better clarify how it learns to manage the resources in the platform.

4.1 Problem Formulation

The *shutdown problem* consists of determining the moments to shutdown resources in order to save energy. Integrating it to an OR approach adds another layer of complexity, requiring the solution to determine for how long a job can be delayed in order to save energy.

Jobs are delayed to mitigate the main disadvantage of pure shutdown policies, when the switching cost is higher than the cost of letting it idle or off. Therefore, the solution reserves some nodes to itself and keep them reserved while there is no expected gain on releasing them for a job. We leverage the power of DRL methods to teach an agent on how to perform this task. This learning phase occurs during the interaction with an environment, which must be defined as a MDP. A MDP is a mathematical framework for decision-making tasks that 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 [47].

The MDP defines the rules that orbits the relation among the agent, the environment, and the task it must perform. Given an state s_t at time t, an agent must choose an action $a_t \in A(s)$ which induces a probability distribution $T(s_t, a_t)$ over S to target states. In the next time step, the agent receives s_{t+1} along with a reward signal $R(s_t, a_t, s_{t+1})$ representing the quality of the action a_t taken at state s_t. This process continues until a terminal state is encountered. The main objective is to select the sequence of actions that maximizes its total expected reward. Thus, the agent optimize its policy $\pi : S \to A$ through the reinforcement of the best rewarded actions for each state. The expected reward is an estimation of the real state/action values learned following a balance between exploration-exploitation. In the exploration phase, the agent collects new experiences that may allow it to overcome some local minima. In the other hand, the exploitation reinforces the best experiences and approximates its estimations based on the values observed in each state. Therefore, an important aspect of DRL methods (including RL) is to correctly estimate the value of the states or action-state pairs.

Following this framework, we formulate the shutdown problem into a MDP as follows.

State Space

We define the state as a function of n past observations of the environment $H_t = [O_{t-n}, ..., O_{t-1}, O_t]$ since time t. An observation O combines the current platform state, the current queue state and the current simulation state. The platform state provides the number of resources in each resource state $[|r_{off}|, ..., |r_{computing}|]$. The queue state provides the number of jobs in the queue $|Q|$, the promise $prom_j$ given by the scheduler policy for the start of the first job j in the queue and the features f of the first k jobs, which are:

$$f_k = [q_k, wall_k, stretch_k, |j_{user}|] \tag{1}$$

$|j_{user}|$ represents the total number of jobs from the same user currently on the system and the *stretch* is the ratio between the waiting time (*wait*) and the expected processing time of a job j, given by:

$$stretch_j = \frac{wait_j}{wall_j} \tag{2}$$

Finally, the simulation state describes the current simulation time. The idea is allow the agent to infer the times in which the system is prone to receive a sequence of submissions.

Action Space

Similar to a malleable job, the agent can expand or shrink its reservation size by requesting nodes to the RJMS. The reservation size is what determines the number of nodes which will be kept reserved to the agent and unavailable for scheduling. Therefore, the action space is given by $\{\theta, 1, ..., G\}$, where $a = G$ means the agent wants to reserve all nodes in the platform; and $a = \theta$ indicates the agent does not wish to reserve any node. The agent represents the platform administrator's perspective.

When the agent makes a reservation the nodes are immediately switched off following the model described in Section 3.1. Reserved nodes cannot be freely used by jobs until the agent decides to decrease its reservation size. This behavior is similar to when a user requests a number of resources but instead of computing a job the resources are switched off.

Reward Function

We craft the reward signal to guide the agent towards our main objective: minimize the energy waste while not degrading performance beyond a threshold. Therefore, the reward is defined by Equation 3.

$$R = -(E_{waste} + QoS) \tag{3}$$

The energy waste (E_{waste}) is defined in Equation 4. It corresponds to the total amount of energy spent by idle and switching nodes since the last decision-making process.

$$E_{waste} = E_{idle} + E_{off \to on} + E_{on \to off} \tag{4}$$

The QoS is a job-centric metric and is defined by Equation 5.

$$QoS = \sum_{j \in Q} \begin{cases} q_j & \text{if } wait_j \geq wall_j * \tau \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The τ parameter controls the aggressiveness of the algorithm by delimiting the maximum desirable waiting time for each job in the queue. The main idea is to encourage the algorithm to delay some jobs expecting it can avoid unnecessary node switching and reduce the waste of energy. In other words, the agent must find opportunities in which the extra delay on some jobs actually pays off. Therefore, the reward can be interpreted as a penalization proportional to the current number of idle resources, the current number of resources switching from states $off \to on$ and $on \to off$ and to the current number of resources requested by jobs waiting in the queue that extrapolates the boundaries defined in the QoS metric.

4.2 Algorithm Description

In the core of grid systems, RJMSs is the main tool for platform and job management. It includes the reservation, allocation, scheduling, launching and monitoring of jobs and resources. These procedures can be designed and implemented as independent components that act coordinated. In this sense, DeepShutdown can be seen as an additional component within a RJMS.

Figure 5 illustrates how our proposed approach is integrated into a general RJMS workflow. The algorithm acts just before the scheduler decides the number of resource candidates to be reserved. The RJMS interprets this requisition in the same way it would interpret users requests and the resources are reserved. In the second step the scheduling algorithm selects the jobs that must be executed on the remaining resources and the process goes on to the next timestep.

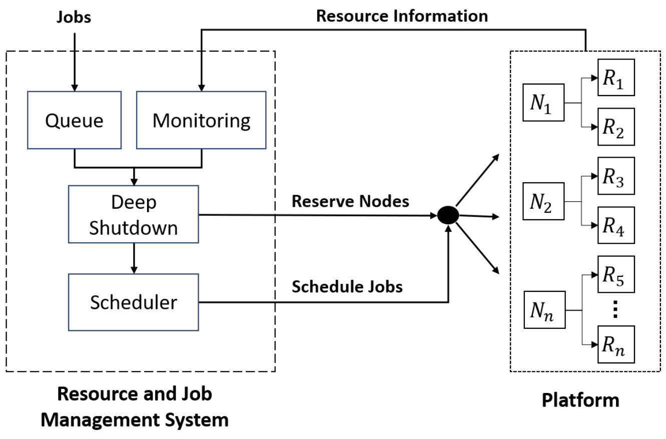


Figure 5: The structure of the DeepShutdown.

By making decisions before the scheduler, DeepShutdown can dynamically define boundaries for the scheduling algorithm by limiting the scheduling space that can be explored to fit the jobs from the queue. The goal is to force the scheduling algorithm to decrease the number of candidates in order to minimize its own objective by predicting the scheduling decisions. Moreover, the algorithm can block scheduling decisions that could potentially undermine its objective.

4.3 Training Algorithm

In this paper we use the Proximal Policy Optimization (PPO) method [44] to train the agent through an actor-critic style. In such style, N parallel actors are responsible for the decision-making while a critic estimates the value of each observed state to evaluate actors decisions. The actors policy and the value function of the critic are represented by artificial neural networks. Besides the hidden layers, the actor network applies a Softmax function in the last layer to output a probability distribution over all possible actions while the critic network applies a linear function. The core idea is to increase the probability of the actions that are better than the expected return estimated by the critic in each observed state. Algorithm **??** shows the pseudocode for the training algorithm using PPO.

Algorithm 1:

```
1: for $iter = 1, 2, ..., I$ do
2:     for $actor = 1, 2, ..., N$ do
3:         Run policy $\pi_{\theta_{old}}$ for $T$ timesteps
4:         Compute advantage estimates $\hat{A}_1, ...\hat{A}_T$
5:     Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
6:     $\theta_{old} \leftarrow \theta$
```

The algorithm instantiate each actor with an independent instance of the environment (see Section 4.1). The actors runs a policy $\pi_{\theta_{old}}$ for T timesteps and the critic estimates the value of each observed state $V(s_t)$ to compute the advantage estimates. The advantage function tells the agent how much better its decision-making was when compared to what is actually known. In order to achieve this end, we use the Generalized Advantage Estimator (GAE) [43] given by Equation 6.

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \tag{6}$$

Where $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, γ is a discount factor, λ adjust the bias-variance trade-off, $V(s_t)$ is an estimate of the value of state s and r_t is the reward received at time t . Then, the surrogate loss $L(\theta)$ is computed and optimized with Adam algorithm [24].

$$L(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t] \tag{7}$$

The PPO method introduces the objective function defined in Equation 7. This function modifies the objective defined in the first term $(r_t(\theta)\hat{A})$ by clipping $r_t(\theta)$ at $1-\epsilon$ or $1+\epsilon$, depending on the advantage $\hat{A}_t$ estimation. Formally defined in Equation (8), the $r_t(\theta)$ gives the probability ratio between the policy with the actual parameter vector θ and the policy with the parameter vector before the update θ_{old}.

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{8}$$

In this way, 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 [44].

5 Evaluation

In this section we describe the evaluation methodology adopted to evaluate the performance of DeepShutdown against commonly used shutdown policies. First, we detail the metrics we consider to than introduce the simulation environment along with the parameters choice. Lastly, the results for each experiment conducted are summarized and presented.

5.1 Metrics

We evaluate the performance of DeepShutdown and its behavior using five performance metrics as follows.

Energy waste

Defined in Equation 4, the total amount of energy wasted has a direct impact on the energy efficiency of each policy. When resources are idle or switching between off and on they are useless for the providers, neither executing a user job or saving energy. Therefore, we consider this amount of energy as a complete waste and its minimization is of interest to grid providers.

Total number of on-off cycles per node

The objective of this metric is to evaluate the aggressiveness of each policy and how much it impacts in the energy consumption. A large number of cycles is not desirable as it can damage the hardware and it can create heat spots when a large number of resources are simultaneously switched on. This is also a metric of interest to providers as users wants their jobs to be executed as soon as possible.

Slowdown

In order to balance this equation, we also consider some job-centric metrics. The first one is the slowdown, which measures the ratio between the time that a job j spent on the system and its actual processing time p_j [15].

The total time a job spent on the system (also called turnaround time) is a function of its waiting time and its processing time. The turnaround time is defined as $turnaround_j = wait_j + p_j$ and the waiting time is defined as $wait_j = start_j - r_j$, in which $start_j$ is the time at which the job started. Given that, the slowdown can formally be defined as given by Equation 9.

$$slowdown_j = \frac{turnaround_j}{p_j} \tag{9}$$

Execution delay

The slowdown can be considered as measure of the fairness of the scheduling policy and may not be fully adequate to analyze an OR policy. Such technique resides on the idea it is possible to delay some jobs in order to save energy. Therefore, the slowdown will likely be greater than or equal to a timeout policy. In order to give a fair comparison, we analyze the extra execution delay $delay_j$. This metric is defined by Equation 10.

$$delay_j = \begin{cases} wait_j - (wall_j * \theta) & \text{if } \frac{wait_j}{wall_j} \geq \theta \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In other words, the $delay_j$ of a job j indicates how much time over a threshold it was forced to stay in the queue. In this case, an OR policy is allowed to delay a job j if its waiting time $wait_j$ is less than the threshold $wall_j * \theta$.

Stretch

Finally, we also consider the stretch of each job. The stretch is the ratio between the waiting time $wait_j$ and the expected processing time $wall_j$. We use this metric due to the fact an OR approach may consider the $wall_j$ of each job to decide if it can be delayed. Moreover, jobs with a small processing time have a high influence on the slowdown and may penalize a policy even if they had an irrelevant waiting time. One of the objectives of an OR approach is to exclusively delay such jobs to save energy. Therefore, we only analyze the slowdown when discussing about the policy learned by the DeepShutdown. For this same reason, we decided to not use the bounded slowdown which diminishes the effect of small jobs (with a processing time of 10 or less minutes) [15].

5.2 Simulation Environment

We simulate the behavior of a RJMS and a grid platform to evaluate the performance achieved by the DeepShutdown strategy and the possibles gains of an OR approach. We build an extension of Batsim[3] [13] to handle the peculiarities of DRL techniques named GridGym. GridGym follows the OpenAi Gym framework [6] and can be easily extended to handle other simulation scenarios. In order to guarantee reproducibility, we provide this extension along with the environments used in a Git repository (see `https://github.com/blindversion`).

We compare the performance of DeepShutdown to three pure timeout policies and one ideal OR technique. The timeout policies turn off the resources following different idling times $t = [0, 1$ min and 5 min$]$. We choose these idling times to analyze how the shutdown aggressiveness can impact the energy consumed. Nevertheless, it allows us to compare the behavior of the policy learned by the DeepShutdown strategy by checking how close it's from a pure shutdown policy.

The last policy is an ideal OR technique that keeps all resources reserved and turned off as long as possible, respecting the Quality of Service (QoS) metric defined in (10). Thus, this policy uses future information about the actual processing times of each job running. Based on this information, it can find out the ideal moments to delay some jobs in the queue. This is an unrealistic policy because the actual processing time cannot be known before the job finishes but it can give some insights about how much we can achieve by following an OR technique instead of a pure timeout policy. This technique will only reduce its reservation size when the jobs cannot be delayed. Moreover, it receives the same view of the system as the other policies.

These policies depict a broad representative sample of shutdown policies deployed on real data centers [40]. In order to provide a fair comparison, we use the same scheduling policy and parameters in all experiments. We use the Smallest Estimated Area First (SAF) policy along with a backfilling mechanism to increase the utilization of the platform by handling jobs in the queue that can immediately start without delaying the first job execution to handle the scheduling task. This policy behavior follows the EASY Backfilling policy, but instead of backfilling jobs in a First In First Out (FIFO) order the queue is sorted by the estimated area of each job in an ascending order. This area is defined by Equation 11 [7]:

$$f(j) = wall_j * q_j \tag{11}$$

In order to evaluate the policies, we use the traces analyzed in Section 2.2. Thus, for a better understanding of the potential gains of an OR approach, we group the traces from each cluster into five bins (25%, 50%, 75%, 100%) based on the total occurrences of sequential jobs in each day. If the workload does not have any sequential jobs, applying an OR approach would not make sense. Therefore, we split the traces into bins to make it possible to evaluate different scenarios and provide a deeper analysis.

Table 2 summarizes this pre-processing step and includes the total number of days (workloads) per group for each cluster trace. Only days which had at least two or more jobs are included and the simulation is done one day at a time following an episodic setting. Each day in the trace correspond to a single workload and simulation experiment.

Table 2 Number of days per group for each cluster trace analyzed.

	Groups				
	[0, 25%) #1	[25, 50%) #2	[50, 75%) #3	[75, 100%) #4	[100%] #5
Econome	259	520	504	271	39
Graphite	487	705	427	149	24
Orion	659	777	496	129	28
Taurus	441	947	612	130	22

In order to speed up the simulation process, we scale up the time in the traces to minutes. Therefore, each experiment correspond to 1440 timesteps.

The platform configuration is based on the hardware of each cluster defined in Table 1. For simplicity, the platform uses the same hardware configuration in every experiment but the number of resources available is different for each cluster setup. The hardware configuration along with the power profile are defined in

Figure 4 and follows the hardware setting of the cluster Taurus [39]. In order to simulate the exact processing time of the jobs in the workloads, we defined each resource can compute 1 Mflop/s.

DPM strategies can be deployed at different levels. In this work, we define that each policy can control the state of the nodes instead of controlling resources individually. This idea follows the default behavior expected when a node is turned off. In this case, all of its resources become unavailable and can only be used after being turned on. This behavior follows the description given in Section 3.1. The number of nodes and its number of resources follows the actual hardware setting of each cluster[4].

5.3 Parameters

We instantiate 16 parallel actors to collect experiences for training parameters. The neural network is composed of two layers. The first one is an LSTM layer with 128 memory units and the second is a FNN with 64 units. The discount factor (γ) is set to 0.99 and λ is 0.95. The ϵ parameter of the objective function (Equation 7) is 0.20. The algorithm is trained for 50 million timesteps and the optimization is done every 1440 timesteps for 4 epochs. This is equivalent to training after a day of experiences.

Our environment parameters included the last 20 observations of the environment into the state and the algorithm can see the first 10 jobs in the queue. The parameter τ of the reward function is 0.50. The simulation time is fixed at 1440 minutes regardless of whether there are jobs to be submitted or to be completed. In order to provide a fair comparison, each policy receives the same snapshot of the environment. After every simulation, the environment is completely rebooted (clean slate).

Moreover, to validate the agent can handle unseen data we also split the traces of each group into two distinct data sets. We reserve 80% of the traces for the training phase and the remaining one are only used for testing. During the training phase, the agent selects a random workload from the training set following an uniform distribution. In this way, we minimize the risks of over-fitting the model to a specific workload.

5.4 Results

Figure 6 plots the normalized cumulative daily energy waste for DeepShutdown *versus* other policies at different workload groups. Each subplot contains a dashed line separating the results for each data set.

The left side gives the results for the training set while on the right side are the results obtained with the testing set. In this way, we can compare if DeepShutdown can keep its performance on new data. Moreover, experiments conducted with DeepShutdown are averaged over 4 repetitions because its policy is not deterministic.

Not surprisingly, the timeout policies are less energy-efficient as the idling time increases. An aggressive shutdown that turns nodes off immediately after they become idle demonstrates considerably energy savings when compared to a soft policy but the same cannot be said when compared with an OR approach. In all traces, the unrealistic OR policy (OR*) achieved best energy savings. The DeepShutdown (DS) exhibits a similar behavior to the OR* policy, and it considerably surpass its results on some traces. This becomes more evident on the traces from clusters Orion and Graphite that are small size clusters and on the Groups 4 and 5 that has a high number of sequential jobs. Group 5 traces shows DS achieved the best energy efficiency but at Groups 1 and 2 from Econome and at Group 3 from Taurus it mimics the behavior of the $T(0)$ policy achieving the same efficiency.

Analyzing the results from Group 5, the DS performance indicates that the agent figured out that it could surpass the QoS metric to achieve even higher energy savings. In this case DS favored the minimization of the energy in detriment of the QoS. This becomes even more evident on results obtained with traces from Orion. In this case, DS is equal to or better than the OR* policy. Moreover, DS achieved good performance regardless of the changing nature of the workloads. Its performance on new data stayed constant, and no performance slowdown was observed.

Comparing the averaged results on a group basis, at Group 1 the OR* policy achieved better energy savings than DS by 2.5% while DS saved 13%, 25.4% and 51% more energy in comparison to pure shutdown policies T(0), T(1) and T(5) respectively. Analysing Group 2, a similar behavior is observed and the OR* policy is better than DS by 12.5%. Group 3, DS is better than the most aggressive shutdown policy T(0) by 10.8% and it saves 44% more energy than the most soft policy T(5). Furthermore, the OR* policy exhibits an improvement of 18% when compared to the DS in this group of jobs.

Group 4, OR* saves just 12% more energy than DS while DS saves 18% more energy than the T(0) policy. Lastly, Group 5 presents DS is better than the OR* policy by 18% while the differences from pure shutdown policies increases up to 26.7%, 34.8% and 56.1% with T(0), T(1) and T(5) respectively.

From another perspective, in Table 3 we average the results in a cluster basis. We observed, again, DeepShutdown stands out in terms of energy efficiency when compared to the timeout policies. Compared to the Timeout (0) policy, DS achieved a reduction of 2.9% to 30% on the waste of energy with about 11.6% to 33.8% less shutdown events. Compared to a soft shutdown policy, DS outperformed the Timeout (5) by 40.6% to 56.5% on the energy waste while the average number of shutdown events is similar. In the Graphite and Orion traces, it even surpassed the Timeout (5) in the number of shutdown events by 11.1% and 8.5% less state switches saving 56.5% and 55.4% more energy respectively.

Compared to the unrealistic OR* policy there are even better possibilities to save energy by delaying some

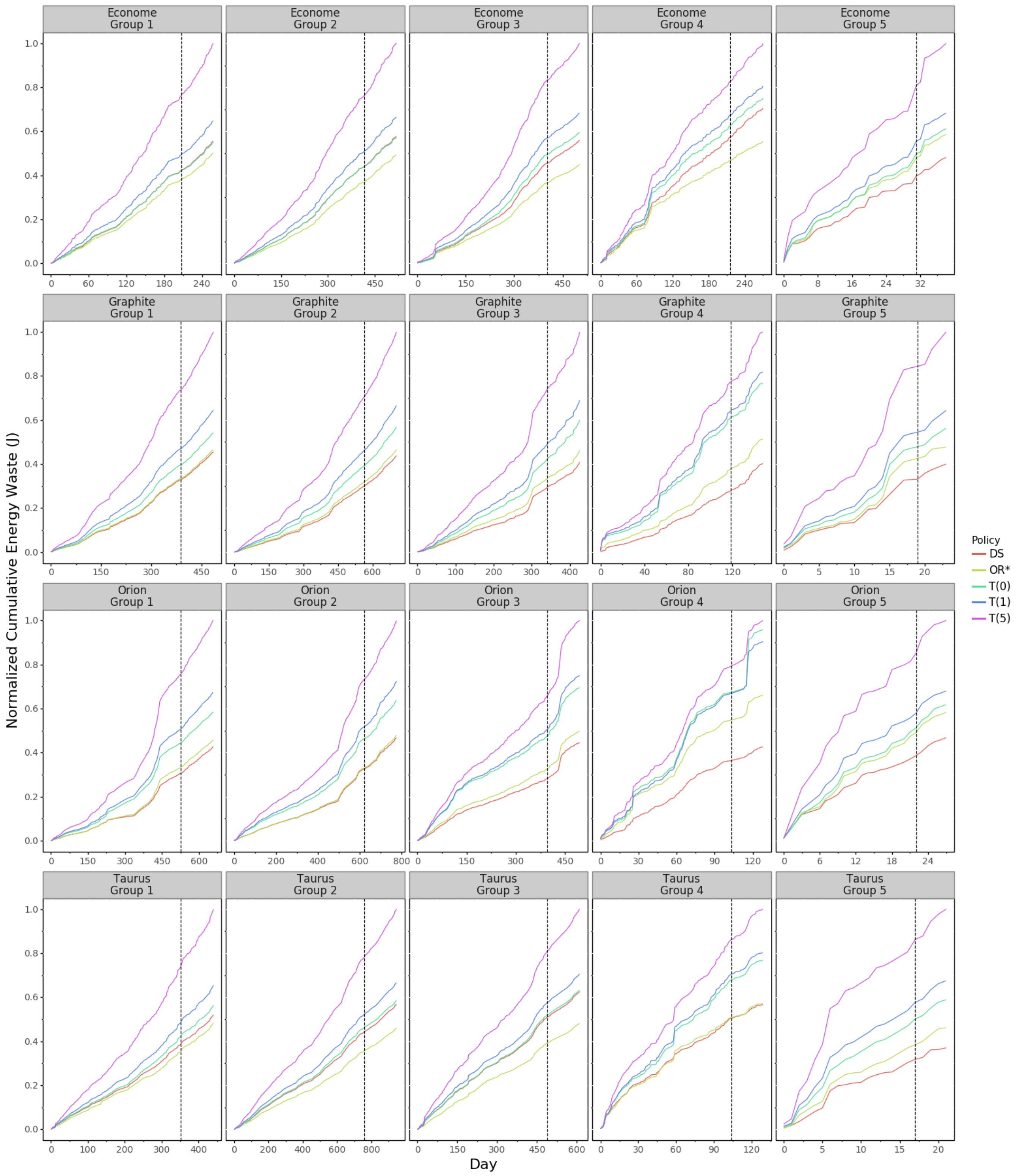


Figure 6: Cumulative daily waste of energy. For each trace, the dashed line defines a boundary between the Training (left side) and Testing days (right side).

jobs execution. The DS was only better at the Graphite and Orion traces, which are the smallest clusters in number of resources. In the remaining traces, the OR* policy saved 20% more energy on average than the policy learned by the DeepShutdown with 10-16,1% less shutdown events. Moreover, it is as softer as the Timeout (5) policy while achieving considerably higher rates of energy savings. These results demonstrate that an OR approach can in fact minimizes the wastes with a similar

Table 3 Overall performance of DS compared against other heuristics over all traces.

		Delay (min)				Energy Waste (J)				# Switches				Stretch (min)			
		avg	std	min	max	avg	std	min	max	avg	std	min	max	avg	std	min	max
Econome	DS	**48.50**	116.77	0.0	1002.00	21738.40	20493.94	125.0	199485.5	93.50	80.38	1.0	797.00	0.22	1.03	0.0	26.68
	OR*	53.16	124.34	0.0	1002.00	**18076.86**	19474.91	125.0	223416.0	**84.96**	86.38	1.0	1044.00	0.21	1.07	0.0	26.68
	T(0)	49.69	123.29	0.0	1002.00	22392.20	23398.09	250.0	271352.0	106.12	106.36	2.0	1268.00	0.15	1.00	0.0	26.68
	T(1)	49.54	123.24	0.0	1002.00	25483.59	25960.63	250.0	329490.0	100.21	97.68	2.0	1260.00	0.15	0.99	0.0	26.68
	T(5)	49.48	123.54	0.0	1002.00	36624.22	32931.17	250.0	307125.0	85.35	72.73	2.0	720.00	**0.14**	0.99	0.0	26.68
Graphite	DS	**50.92**	116.47	0.0	770.00	**6712.60**	6219.47	125.0	51549.5	**30.63**	27.94	1.0	238.00	0.41	1.38	0.0	22.88
	OR*	54.27	123.59	0.0	770.00	7216.18	6587.65	125.0	72855.0	33.92	30.36	1.0	340.00	0.30	1.39	0.0	29.28
	T(0)	52.81	125.29	0.0	770.00	8950.24	8287.19	125.0	89024.0	42.20	38.47	1.0	416.00	0.26	1.41	0.0	29.28
	T(1)	52.99	125.43	0.0	770.00	10401.94	9381.74	125.0	87820.0	40.35	35.84	1.0	350.00	0.26	1.40	0.0	29.28
	T(5)	52.91	125.40	0.0	770.00	15434.66	13357.23	125.0	87452.0	34.48	28.82	1.0	208.00	**0.25**	1.41	0.0	29.35
Orion	DS	72.27	132.62	0.0	782.67	**6284.59**	6930.93	125.0	70165.0	**28.07**	30.69	1.0	291.25	0.54	2.10	0.0	66.24
	OR*	70.34	136.79	0.0	782.67	6763.39	7551.15	125.0	67003.0	31.84	34.79	1.0	302.00	0.41	2.04	0.0	63.95
	T(0)	67.23	139.61	0.0	782.67	8996.35	9892.78	250.0	146727.0	42.41	45.92	2.0	687.00	**0.30**	1.47	0.0	34.19
	T(1)	**67.14**	139.48	0.0	782.67	10081.97	10864.42	220.0	108591.0	38.95	40.81	1.0	356.00	**0.30**	1.47	0.0	34.19
	T(5)	67.39	139.76	0.0	782.67	14104.64	15307.39	220.0	125931.0	30.69	33.36	1.0	295.00	**0.30**	1.47	0.0	34.19
Taurus	DS	**48.31**	106.29	0.0	1149.00	16513.70	12811.54	125.0	134672.0	72.53	54.15	1.0	584.00	0.30	1.48	0.0	28.54
	OR*	53.60	113.04	0.0	1149.00	**13747.39**	12307.84	125.0	194121.0	**62.43**	53.19	1.0	904.00	0.29	1.46	0.0	29.51
	T(0)	49.99	114.82	0.0	1149.00	17441.70	14984.53	375.0	264884.0	82.10	68.27	3.0	1236.00	**0.22**	1.45	0.0	29.51
	T(1)	50.02	114.75	0.0	1149.00	19664.81	16836.47	375.0	271768.0	76.56	63.23	3.0	1032.00	**0.22**	1.44	0.0	28.72
	T(5)	50.23	115.25	0.0	1149.00	28714.38	22920.56	375.0	256368.0	65.29	49.51	2.0	612.00	**0.22**	1.44	0.0	29.12

behavior to a soft shutdown policy if we allow it to delay the execution of some jobs.

By observing the job-centric metrics, we notice that DeepShutdown is more aggressive than the OR* policy. In the worst case it increased the averaged stretch time by 36.6% but it remains below the defined threshold (see Section 5.3) in almost all traces. Compared to the Timeout (0), DeepShutdown increased by 46.6% to 80% the averaged stretch time. These results demonstrates the trade-off between energy savings and performance: DeepShutdown increased the stretch time to increase the energy savings.

Not surprisingly, all timeout policies presents a similar stretch time because they do not deliberately delay the jobs. The minor differences observed are due to the time required to turn on and off the nodes. This behavior can also be seen on the delay time, there is a very slight different between these policies. In almost all traces the averaged delay time is smaller on the DeepShutdown results. It was better than the OR* policy, which indicates that its strategy did not force the jobs to wait for too long periods after violating the defined QoS. In other words, it is trying to minimize the downside effect of an OR approach.

The standard deviation and the range of the values are considerably high. Furthermore, the values observed for each trace are very different and cannot be compared. This means the workloads can drastically change within distinct clusters and days, reinforcing what was already observed on [26].

6 Discussion

This section analyzes what DeepShutdown has learned. We present the characteristics of the most delayed jobs and compare them to the OR and Timeout (0) policies. Moreover, we provide information about the convergence behavior of DeepShutdown.

6.1 What is DeepShutdown doing?

The idea of an OR approach is to explore the workload properties in order to save energy by delaying some jobs. Specifically, we want the algorithm to delay the sequential jobs which may cause unnecessary boot ups. In order to validate this idea, we first analyze the slowdown of the jobs as function of its interarrival time. Figure 7 shows the slowdown of each trace on distinct groups of jobs.

A logarithmic transformation was performed on the slowdown to reduce the effect of outliers and the jobs are grouped based on their interarrival time. DeepShutdown is compared to OR* to check how close its behavior is from a OR approach. Due to the same reason, we compare it with the Timeout (0) to check how close it is from a pure shutdown strategy. We can noted DeepShutdown exhibits almost the highest range of slowdown values in every job group. The slowdown is considerably higher on jobs with a small interarrival time but it eventually diminishes with the increase in the interarrival time. This means that the DeepShutdown is indeed favoring the delay of sequential jobs, so its behavior is closer to an OR approach. The same can also be observed for the OR* policy, which validates this insight.

Just delaying the sequential job is inefficient to guarantee higher energy savings. On one hand, if the job executes for a long period, then there is no gain in forcing the delay of the next job in the queue. On the other hand, if it exhibits a small processing time than the extra delay may pays off. We analyzed the actual processing time of the most delayed jobs on each trace. The resulting analysis is presented in Figure 8. The actual processing time is normalized by a logarithmic transformation.

DeepShutdown is mostly delaying the jobs with a small processing time when compared to the Timeout (0) policy. This behavior becomes more evident on the Orion traces that exhibits the best energy savings achieved by DS. Delaying jobs with small processing

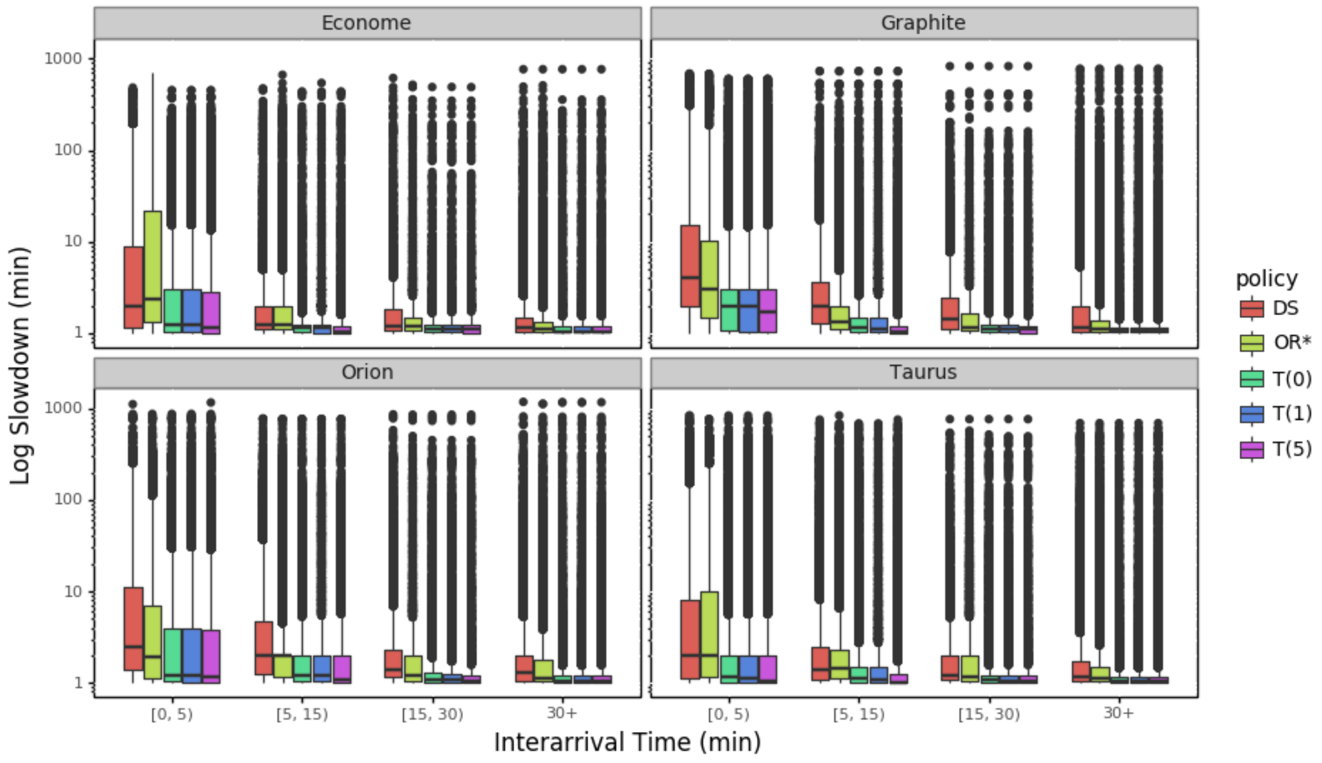


Figure 7: Slowdown comparative of distinct job groups on each trace.

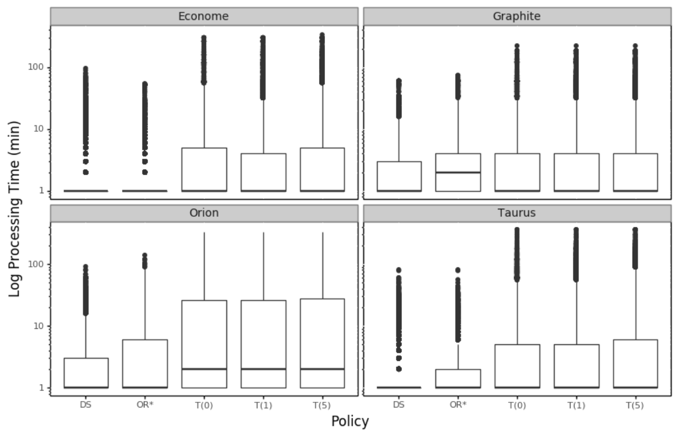


Figure 8: Current processing time of the top 10k jobs with highest slowdown values.

times minimize the number of boot ups while increasing the energy savings. This indicates that DeepShutdown is indeed exploring the workload properties to identify the sequential jobs with small processing times.

6.2 The Convergence Behavior

We analyze the performance of DeepShutdown during the training phase to understand its convergence behavior. Figure 9 illustrates the learning curves for each trace. Each value is an average of 100 experiments of all workloads from the training set and the score is defined in Equation 3. We showed the averaged scores of the policies with the highest energy savings to compare performance over time along with a policy that reserve nodes by random.

As expected, the performance of DeepShutdown improves with the number of iterations. On the beginning it shows very low performance and its behavior is similar to the random policy. When it starts to interact the environment its performance starts to improve. On the smallest clusters, in number of resources (Graphite and Orion), DeepShutdown exhibits the best performance after 100-250 iterations. On ther other hand, with bigger platform sizes (Econome and Taurus) it took more iterations to learn a policy which give results close to the OR* policy. This happens due to the fact of the state space increasing with the platform size, therefore the exploration is faster on cluster with a small number of resources.

The same variation observed on Table 3 is also observed in the learning curves by analyzing the variation of the scores. This indicates that identifying sequential jobs with small processing times is not a straightforward task. Besides exhibiting a similar frequency of sequential jobs, each experiment (called episode in a RL setting) considerably differs from each other.

7 Related Work

The power efficiency of computing platforms started to become a concern in the 2000s. The performance-at-any-

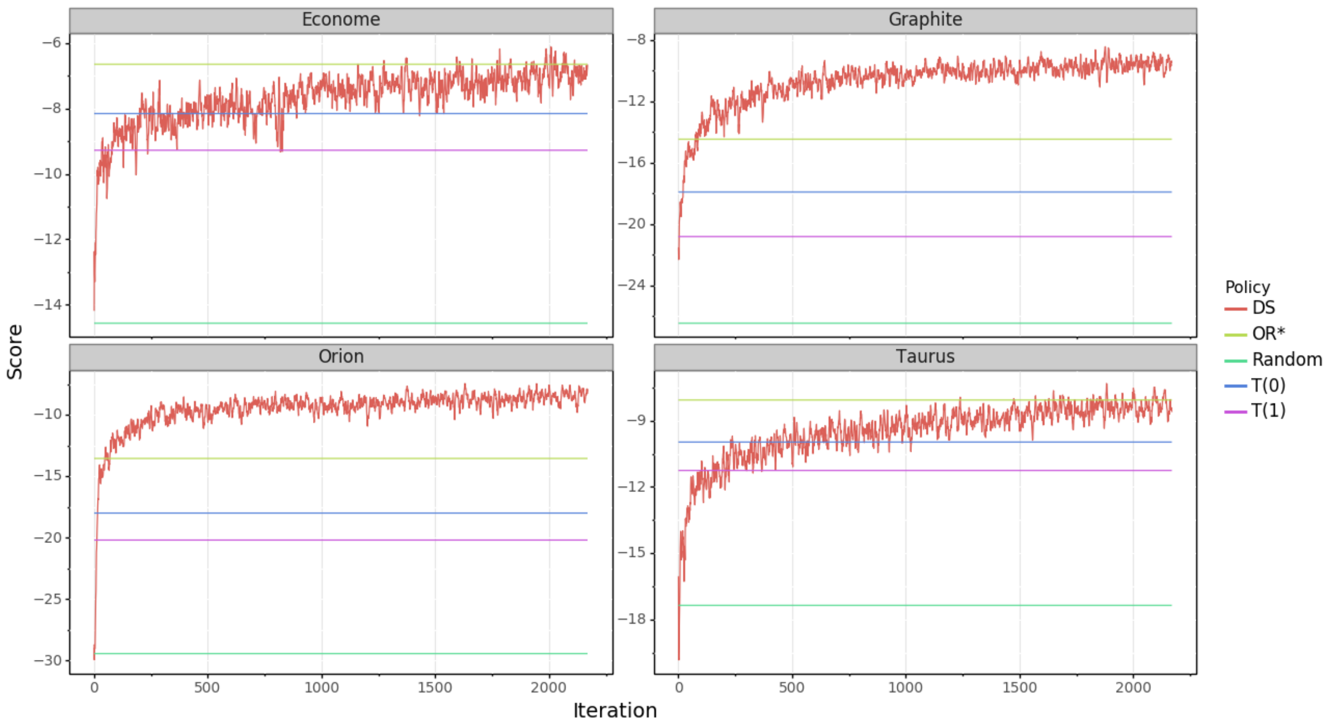


Figure 9: Learning curve.

cost paradigm is neither sustainable nor environmentally friendly [17]. Since then there is a broad range of studies focusing on different strategies. A few namely: fine-grained power management [14, 29]; coarse-grained power management [12]; job scheduling [16]; and thermal management [42].

Earlier studies on shutdown strategies started on 2001. Using a load distribution algorithm it was possible to save energy by concentrating the load on fewer nodes and switching-off the remaining ones [38]. In a similar way, Muse [9] uses an economic framework and a greedy algorithm to dynamically adapt the number of active resources to the demand. It concentrates the load on the minimal active set of resources in order to save energy. Both approaches are similar to ours since they adapt the number of active resources based on the load to increase the possibilities in energy savings. However, they did not consider the transition costs for switching resources between on and off.

Considering the transition cost, is important due to the time and energy required for switching a resource. Moreover, a resource cannot be used while switching between states. Thus, from this perspective, ERIDIS [35] works at the platform level and decides whether a resource must be turned off based on workload predictions. The prediction part relies on averaged values of past inactivity periods and feedback given by the differences observed from the predictions and the real values. In a similar way, the Inertial Shutdown algorithm [39, p. 85] adopts an OR approach to dynamically adjust the number of active resources based on estimations of the unresponsiveness variation. This unresponsiveness is an estimation of the required amount of time to compute the pending load in the queue. When the unresponsiveness is increasing the algorithm switches some resources on otherwise it will turn them off. Both approaches use predictions to decides when resources must be turned off. The main difference to our approach is that we use DRL to train an agent to deal with the shutdown of resources. The prediction part is done at the agent and is inferred from the experiences observed during the training phase.

Several other studies used RL for resource management [20, 31, 51, 53, 54]. Moreover, DRL is an extension of RL methods that uses DL methods to deal with complex tasks. Its main adoption comes from the recent breakthroughs achieved with DRL techniques [30] leading to questions about its performance when dealing with resource management problems. With this in mind, DeepRM [28] is an attempt to build agents that learn to schedule in order to minimize the slowdown. It uses a similar method called REINFORCE but its objective and environment model (MDP formulation) differ from ours. Exemplifying, it summarizes the platform by using a matrix of RxT, where R is the number of resources and T is the time window. The idea is similar to a Gantt chart and each job in the queue is also represented by this matrix. Moreover, the evaluation is conducted using synthetic workloads while we use traces from a real grid system. DRL-Cloud [10] is another DRL approach to minimize the energy cost in cloud computing. It uses a value-based method named Deep Q-Networks (DQNs) to deal with the resource provisioning and task scheduling. Both cloud workloads and environment model greatly differs from the models we adopted in this work. In grid platforms, a node is commonly not allowable to be shared among different users while in cloud computing virtual machines from distinct users can be hosted on the same node. Therefore, further comparisons cannot be made.

Liu et al. [27] proposed a hierarchical approach combining RL methods in different levels to deal with the resource allocation and the power management. At the highest level it uses an auto-encoder to extract a lower-dimensional input representation and a DQN to allocate resources. In the local tier an Long Short-Term Memory (LSTM) network is trained to predict the next job inter-arrival time to be used by a Q-learning agent in the control of local servers. The main difference is that they combined the prediction part with a RL agent to determine the idling time before a resource is turned off while we adopted an end-to-end solution with DRL. Moreover, our proposal controls a computing platform instead of just a local server.

8 Considerations & Future work

Energy consumption is a key metric related to the sustainability of an da center infrastructure. The increasing in size and complexity of computing platforms requires sophisticated solutions to improve resource utilization efficiency. Performance-at-any-cost is no longer wanted. Dynamic power management procedures take advantage of periods when resources are underutilized or unused to save energy. Idle resources represent a waste of energy since they are neither serving the users nor the providers. Such waste cannot be neglected and different strategies must be deployed to decrease the operational costs.

This paper explores a suite of shutdown strategies. We clarify the main disadvantage of deploying a pure shutdown policy and we propose an alternative that employs an OR approach. This allows the solution to exploit the workload to identify jobs that can be delayed in order to save energy. Keeping this in mind, we leverage the power of DRL to teach an agent how to perform this tasks. The proposed method, named **DeepShutdown**, was able to learn how and when to reserve some resources and turn them off in order to increase the energy savings. Results revealed it had a similar behavior when compared to an oracle-based OR policy. Jobs with small processing times are the most delayed ones. The energy savings become more evident when compared with different rule-based shutdown policies.

Motivated by the lack of such tools we developed a suite of environments which can be used to train agents with RL methods on different resource management tasks. The environment, named GridGym, is an extension of Batsim that leverages the OpenAI framework to handle the peculiarities of such methods. Although there is a good range of rule-based solutions available, there is still room to be explored by adaptive solutions on the exploitation of the workloads patterns for better efficiency. GridGym is a step forward that can facilitate the experiments and the training process.

Applying DRL on resource management procedures is feasible but there is still work to be done. First, we must conduct experiments on traces from others grid platforms. We showed that the proportion of sequential jobs is considerably high on traces observed from the GRID'5000, but questions remain if this behavior can also be seen on other platforms. Another point to be explored is the development of the reward function. Different metrics can also be considered to guide the algorithm. Finally, we must integrate the scheduling problem onto the DeepShutdown environment to verify if it can even surpass the performance achieved when using an external scheduling policy.

Acknowledgement

Blind version

References

[1] Das-4: The distributed asci supercomputer 4. https://www.cs.vu.nl/das4/home.shtml

[2] Bates, N., Ghatikar, G., Abdulla, G., Koenig, G.A., Bhalachandra, S., Sheikhalishahi, M., Patki, T., Rountree, B., Poole, S.: Electrical grid and supercomputing centers: An investigative analysis of emerging opportunities and challenges. Informatik-Spektrum **38**(2), 111–127 (2015)

[3] Benoit, A., Lefèvre, L., Orgerie, A.C., Raïs, I.: Reducing the energy consumption of large-scale computing systems through combined shutdown policies with multiple constraints. The International Journal of High Performance Computing Applications **32**(1), 176–188 (2018). DOI 10.1177/1094342017714530. URL https://doi.org/10.1177/1094342017714530

[4] Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, 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). DOI 10.1177/1094342006070078. URL https://doi.org/10.1177/1094342006070078

[5] Borghesi, A., Collina, F., Lombardi, M., Milano, M., Benini, L.: Power capping in high performance computing systems. In: G. Pesant (ed.) Principles and Practice of Constraint Programming, pp. 524–540. Springer International Publishing, Cham (2015)

[6] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)

[7] 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)

[8] 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)

[9] Chase, J.S., Anderson, D.C., Thakar, P.N., Vahdat, A.M., Doyle, R.P.: Managing energy and server resources in hosting centers. ACM SIGOPS operating systems review **35**(5), 103–116 (2001)

[10] 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)

[11] Dayarathna, M., Wen, Y., Fan, R.: Data center energy consumption modeling: A survey. IEEE Communications Surveys & Tutorials **18**(1), 732–794 (2016)

[12] Dutot, P., Georgiou, Y., Glesser, D., Lefevre, L., Poquet, M., Rais, I.: Towards energy budget control in hpc. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 381–390 (2017). DOI 10.1109/CCGRID.2017.16

[13] 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)

[14] Etinski, M., Corbalan, J., Labarta, J., Valero, M.: Understanding the future of energy-performance trade-off via dvfs in hpc environments. Journal of Parallel and Distributed Computing **72**(4), 579 – 590 (2012). DOI https://doi.org/10.1016/j.jpdc.2012.01.006. URL `http://www.sciencedirect.com/science/article/pii/S0743731512000172`

[15] Feitelson, D.G., Rudolph, L.: Metrics and benchmarking for parallel job scheduling. In: Workshop on Job Scheduling Strategies for Parallel Processing, pp. 1–24. Springer (1998)

[16] Feller, E., Rilling, L., Morin, C.: Energy-aware ant colony based workload placement in clouds. In: Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing, GRID '11, pp. 26–33. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/Grid.2011.13. URL `http://dx.doi.org/10.1109/Grid.2011.13`

[17] Feng, W., Cameron, K.: The green500 list: Encouraging sustainable supercomputing. Computer **40**(12), 50–55 (2007). DOI 10.1109/MC.2007.445

[18] Foster, I., Zhao, Y., Raicu, I., Lu, S.: Cloud computing and grid computing 360-degree compared. In: Grid Computing Environments Workshop, 2008. GCE'08, pp. 1–10. Ieee (2008)

[19] Galizia, A., Quarati, A.: Job allocation strategies for energy-aware and efficient grid infrastructures. Journal of Systems and Software **85**(7), 1588 – 1606 (2012). DOI https://doi.org/10.1016/j.jss.2012.01.050. URL `http://www.sciencedirect.com/science/article/pii/S0164121212000362`. Software Ecosystems

[20] Galstyan, A., Czajkowski, K., Lerman, K.: Resource allocation in the grid using reinforcement learning. In: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 3, pp. 1314–1315. IEEE Computer Society (2004)

[21] Hikita, J., Hirano, A., Nakashima, H.: Saving 200kw and 200 k/year by power-aware job/machine scheduling. In: 2008 ieee international symposium on parallel and distributed processing, pp. 1–8. IEEE (2008)

[22] Hinz, M., Koslovski, G.P., Miers, C.C., Pilla, L.L., Pillon, M.A.: A cost model for iaas clouds based on virtual machine energy consumption. Journal of Grid Computing **16**(3), 493–512 (2018). DOI 10.1007/s10723-018-9440-8. URL `https://doi.org/10.1007/s10723-018-9440-8`

[23] Huang, S., Feng, W.: Energy-efficient cluster computing via accurate workload characterization. In: 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 68–75 (2009). DOI 10.1109/CCGRID.2009.88

[24] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)

[25] 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). DOI https://doi.org/10.1016/j.engappai.2019.02.013. URL `http://www.sciencedirect.com/science/article/pii/S0952197619300351`

[26] 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). DOI 10.1109/IPDPS.2019.00077

[27] Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., Tang, J., Wang, Y.: A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In: Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on, pp. 372–382. IEEE (2017)

[28] 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)

[29] Marzolla, M., Mirandola, R.: Dynamic power management for qos-aware applications. Sustainable Computing: Informatics and Systems **3**(4), 231 – 248 (2013). DOI https://doi.org/10.1016/j.suscom.2013.02.001. URL `http://www.sciencedirect.com/science/article/pii/S2210537913000206`

[30] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)

[31] 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)

[32] Nesmachnow, S., Dorronsoro, B., Pecero, J.E., Bouvry, P.: Energy-aware scheduling on multicore heterogeneous grid computing systems. Journal of Grid Computing **11**(4), 653–680 (2013). DOI 10.1007/s10723-013-9258-3. URL `https://doi.org/10.1007/s10723-013-9258-3`

[33] Orgerie, A., Lefèvre, 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

[34] 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)

[35] Orgerie, A.C., Lefèvre, L.: ERIDIS: Energy-efficient Reservation Infrastructure for large-scale DIstributed Systems. Parallel Processing Letters **21**(2), 133–154 (2011). URL `https://hal.inria.fr/ensl-00618594`

[36] 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). DOI https://doi.org/10.1016/j.jpdc.2017.05.001. URL `http://www.sciencedirect.com/science/article/pii/S0743731517301521`

[37] 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)

[38] Pinheiro, E., Bianchini, R., Carrera, E.V., Heath, T.: Load balancing and unbalancing for power and performance in cluster-based systems (2001)

[39] Poquet, M.: Simulation approach for resource management. Theses, Université Grenoble Alpes (2017). URL `https://tel.archives-ouvertes.fr/tel-01757245`

[40] Raïs, I., Orgerie, A.C., Quinson, M.: Impact of shutdown techniques for energy-efficient cloud data centers. In: J. Carretero, J. Garcia-Blas, R.K. Ko, P. Mueller, K. Nakano (eds.) Algorithms and Architectures for Parallel Processing, pp. 203–210. Springer International Publishing, Cham (2016)

[41] Raïs, I., Orgerie, A.C., Quinson, M., Lefèvre, L.: Quantifying the impact of shutdown techniques for energy-efficient data centers. Concurrency and Computation: Practice and Experience **30**(17), e4471 (2018). DOI 10.1002/cpe.4471. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4471`. E4471 cpe.4471

[42] Sarood, O., Kale, L.V.: A 'cool' load balancer for parallel applications. In: SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–11 (2011)

[43] Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015)

[44] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)

[45] Shi, L., Zhang, Z., Robertazzi, T.: Energy-aware scheduling of embarrassingly parallel jobs and resource allocation in cloud. IEEE Transactions on Parallel and Distributed Systems **28**(6), 1607–1620 (2017). DOI 10.1109/TPDS.2016.2625254

[46] Sun, D., Zhang, G., Yang, S., Zheng, W., Khan, S.U., Li, K.: Re-stream: Real-time and energy-efficient resource scheduling in big data stream computing environments. Information Sciences **319**, 92 – 112 (2015). DOI https://doi.org/10.1016/j.ins.2015.03.027. URL http://www.sciencedirect.com/science/article/pii/S0020025515001929. Energy Efficient Data, Services and Memory Management in Big Data Information Systems

[47] Sutton, R.S., Barto, A.G., Williams, R.J.: Reinforcement learning is direct adaptive optimal control. IEEE Control Systems **12**(2), 19–22 (1992)

[48] Tarplee, K.M., Friese, R., Maciejewski, A.A., Siegel, H.J., Chong, E.K.P.: Energy and makespan tradeoffs in heterogeneous computing systems using efficient linear programming techniques. IEEE Transactions on Parallel and Distributed Systems **27**(6), 1633–1646 (2016). DOI 10.1109/TPDS.2015.2456020

[49] Terzopoulos, G., Karatza, H.: Performance evaluation and energy consumption of a real-time heterogeneous grid system using dvs and dpm. Simulation Modelling Practice and Theory **36**, 33 – 43 (2013). DOI https://doi.org/10.1016/j.simpat.2013.04.006. URL http://www.sciencedirect.com/science/article/pii/S1569190X13000695

[50] 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)

[51] Wu, J., Xu, X., Zhang, P., Liu, C.: A novel multi-agent reinforcement learning approach for job scheduling in grid computing. Future Generation Computer Systems **27**(5), 430–439 (2011)

[52] Young, B.D., Apodaca, J., Briceño, L.D., Smith, J., Pasricha, S., Maciejewski, A.A., Siegel, H.J., Khemka, B., Bahirat, S., Ramirez, A., Zou, Y.: Deadline and energy constrained dynamic resource allocation in a heterogeneous computing environment. The Journal of Supercomputing **63**(2), 326–347 (2013). DOI 10.1007/s11227-012-0740-7. URL https://doi.org/10.1007/s11227-012-0740-7

[53] Zhang, W., Dietterich, T.G.: A reinforcement learning approach to job-shop scheduling. In: IJCAI, vol. 95, pp. 1114–1120. Citeseer (1995)

[54] Zomaya, A.Y., Clements, M., Olariu, S.: A framework for reinforcement-based scheduling in parallel processor systems. IEEE transactions on parallel and distributed systems **9**(3), 249–260 (1998)