Task Scheduler for Heterogeneous Data Centres Based on Deep Reinforcement Learning

Jaime Fomperosa, Mario Ibañez, Esteban Stafford(✉), and Jose Luis Bosque

Dpto. de Ingeniería Informática y Electrónica, Universidad de Cantabria, Santander, Spain
{jaime.fomperosa,mario.ibanez,esteban.stafford, joseluis.bosque}@unican.es

Abstract. This article advocates for the leveraging of machine learning to develop a workload manager that will improve the efficiency of modern data centres. The proposals stem from an existing tool that allows training deep reinforcement agents for this purpose. However, it incorporates several major improvements. It confers the ability to model heterogeneous data centres and then it proposes a novel learning agent that can not only choose the most adequate job for scheduling, but also determines the best compute resources for its execution. The evaluation experiments compare the performance of this learning agent against well known heuristic algorithms, revealing that the former is capable of improving the scheduling.

Keywords: Deep Reinforcement Learning · Task scheduling · Heterogeneous data centres · Machine Learning

1 Introduction

Modern Information Technology (IT) relies heavily on *data centres* which host massive amounts of interconnected computers. A subset of these data centres support the scientific and engineering communities with high performance computing services. The computers that integrate these combine their processing capabilities to accelerate the execution of complex problems [4].

To harness the power of computer clusters, data centres rely on a Workload Manager. It is in charge of job scheduling, or choosing jobs awaiting execution and assigning them to computing resources of the data centre. But this is an NP-Complete problem that cannot be solved in polynomial time. This is exacerbated by the huge growth of data centres [1], the wide variety and heterogeneity of architectures and configurations they host [13,14]. This means that the decision space of the workload manager has increased substantially, and consequently so has the difficulty of finding optimal solutions to the problem. It is possible to find *near-optimal* solutions using *approximation* methods [17] or *heuristic* [11] algorithms. The latter are commonly found at the core of modern resource managers, like Slurm [18]. They are characterised by sacrificing optimality for speed, which is a necessary compromise.

R. Wyrzykowski et al. (Eds.): PPAM 2022, LNCS 13826, pp. 237–248, 2023.
https://doi.org/10.1007/978-3-031-30442-2_18

These heuristic algorithms are fairly simple. Three algorithms are usually implemented nowadays: First In, First Out (FIFO), Shortest Job First (SJF) [12] and BackFill [7]. There are more complex algorithms that consider several attributes of each job in order to compute a score, which is then used to sort and prioritize them, such as WFP3 or UNICEP [16] or F1 [2]. However, these have difficulties in adapting to changes in the resources, the type of job or the objectives. Recently, machine learning has shown its adaptability to different scenarios, contrasting with the static approach of heuristic algorithms [9,10].

Reinforcement Learning (RL) is a branch of machine learning that can autonomously improve its behaviour through trial and error. A key advantage of this approach is that it can consider many more parameters than heuristic algorithms and learn which are the most important. In this context *IRMaSim* [6], emerges as a tool to develop and test reinforcement learning algorithms on a simulator of heterogeneous data centres. A further development of this idea is *RLScheduler* [19]. Its results are fairly good despite its coarse simulator, where only homogeneous data centers with identical compute devices can be modeled.

The main *hypothesis* of this article is that the Deep Reinforcement Learning (DRL) techniques used in [19] can be adapted to schedule jobs in a heterogeneous data centre and with better performance than state-of-the-art heuristic algorithms. The pursuit of this hypothesis requires the completion of three steps. First, the definition of an environment that adequately represents heterogeneous data centres. To this end the cores of the cluster are grouped into nodes with possibly differing properties. Second, is the development of the agent itself, deciding its internal structure, how is the information from the environment fed to it, and how is the action selected. And third, an evaluation procedure must be devised where the performance of the agent is compared to that of well known heuristic algorithms.

The experimental results presented in the evaluation section show two important conclusions. First, that heterogeneity poses new challenges to the scheduling problem, even for classic algorithms that are optimal in homogeneous systems. Secondly, the proposed agent is able to obtain better results in all the studied objectives than heuristic algorithms, which confirms that machine learning-based scheduling is an important new field of study.

The remainder of this article is organised as follows. Section 2 gives an overview of reinforcement learning resource managers. Section 3 describes the main proposals of the article. Section 4 presents the evaluation methodology and discusses its results. Finally, a summary with the most important conclusions of the article is in Sect. 5.

2 Background

Reinforcement learning systems usually revolve around the concept of an *agent* that must drive the behaviour of the *environment* in order to reach a given *objective*. The agent is in charge of making decisions that affect the environment in some manner, and its aim is to learn how to satisfy the objective. Internally the agent is implemented with a *Deep Neural Network (DNN)* that, before going

into production, must be trained. This is done by exposing the environment to stimuli, the agent considers the consequences of the *actions* it takes and it progressively learns which ones are better than others.

The training process is divided in *epochs*, or iterations of sets of stimuli. In turn, epochs consist of a series of *steps*, representing the processing of a given stimulus [15]. In each step the agent performs an *action* that has an impact in the environment. This is measured through *observations* and qualified by a *reward* value that indicates whether the impact was positive or negative. At the end of each epoch, the agent evaluates these and encourages those actions that helped in reaching the objective. After experiencing a number of epochs, the agent converges to using a particular set of actions that maximise the rewards it obtains, and therefore, satisfies the objective.

In the context of resource managers, the environment represents the compute resources of a data centre and the set of jobs, or workload, to be executed. The agent must observe the incoming jobs and the state of the data centre, and decide which job is allocated to which resource in order to achieve an optimization objective, e.g. slowdown or average waiting time. The jobs are usually stored in a *workload queue*, which can potentially be very long and become unmanageable. Modern resource managers, use an *eligible job queue*, which is a fixed length queue that holds the oldest jobs pending execution. The scheduler only considers jobs in this queue for execution, and when one gets chosen, it vacates the queue leaving space for another from the workload queue.

RLScheduler combines a reinforcement learning resource manager with a simplistic data centre simulator to accelerate the training process [19]. The simulated environment defines a number of computational resources, all with the same characteristics. Then it is only necessary to keep a number of free resources to represent the status of the data centre. And knowing to which processors in particular the job is assigned does not really matter. In RLScheduler, an observation represents the state of the environment by means of a vector that contains the attributes of all the eligible jobs.

The simplicity of RLScheduler is also its major drawback, as it considers the resources to be identical and unrelated. On contrast, modern data centres are heterogeneous and structured, as they host compute nodes with a number of processors or cores. These can have different architectures and compute capacities, which can have a great impact on scheduling. In addition, some applications must execute on processors belonging to the same node. Rising to these challenges is the main objective of this paper. Thus, a redesign of RLScheduler is proposed that will allow the modeling of heterogeneous systems. As a consequence, it will train agents to decide on which job to schedule and to which resource it will be assigned.

3 DRL for Scheduling in Heterogeneous Data Centres

This section details the improvements made to RLScheduler allowing its use in heterogeneous and structured data centres, and also proposes a scheduler agent that is able to select the best possible combinations of job and node. In order to

adequately model these systems, the simulated environment must keep track of the jobs assigned to each node and their attributes to properly predict the execution time of the jobs. This also increases the amount of information that must be taken into account by the agent to make the best possible scheduling decisions. As a consequence, the observation and actions spaces must be redefined.

3.1 Observation and Action Spaces

The observation space must be able to represent the state of the environment that the agent will use to decide its next action. Similarly, the action space contains all the possible actions an agent can take over the environment. As mentioned in the previous section, RLScheduler considers all the computational resources of the data centre to have the same properties, making it unnecessary to identify which resources are allocated to each job. However, in heterogeneous data centres it is imperative that the compute resources are represented as a set of nodes with different number of processors. This information must be included in the observation space. Therefore, it is divided in two sets of attributes, the *Node Observation* representing the state of the data centre nodes, and the *Job Observation* containing the job information.

The proposed representation of the computational resources is based on the concept of *node*. Each one can have a different *size* and *speed*, regarding the number of processors it contains and its clock frequency. As for the jobs, they are considered memory-sharing embarrassingly parallel applications requesting a number of *processors*. Meaning that a job cannot be assigned to more than one node, that the node must have enough free processors to host the complete job, and that there is no communication overhead. The reason behind this decision is to streamline the simulator model. The proposed set of attributes for the observation space is shown in Table 1. The number of attributes is lower than in real resource managers, but since the model is expandable, it is fairly simple to add new attributes for the agent to consider.

Table 1. List of attributes of the Job and Node Observations.

Field Name	Notation	Description
Job Observation Space		
Requested Processors	n_j	Number of processors requested for the job
Requested Time	r_j	Amount of time requested for the job
Wait Time	w_j	Amount of time spent by the job in the job queue
Node Observation Space		
Total Processors	tp_n	Number of processors in the node
Free Processors	fp_n	Free processors in the node
Frequency	f_n	CPU clock rate of the node processors

The action space has also been improved to accommodate the node concept. The agent must not only decide which is the next job to be scheduled, but also

to which resource it is allocated. This translates into a new bidimensional action space, where one dimension covers the jobs in the eligible job queue and the other represents the nodes in the data centre.

3.2 Agent Architecture

The proposed agent adheres to the actor-critic architecture, which is common in DRL systems. It combines the use of two similar networks, the actor decides on the next action, while the critic evaluates the performance of the actor. This structure tends to improve training times. Compared to the agent in [19], there are changes in the observation and action spaces that have a significant impact on its design. A diagram describing the new agent, as well as its relationship with said spaces is shown in Fig. 1.

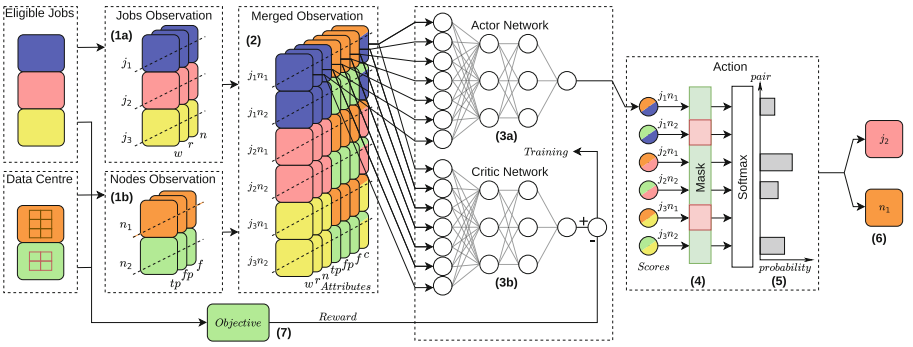

Fig. 1. Proposed agent with observation and actions spaces for three jobs and two nodes.

Since the input to the DNN has to have a fixed size and the number of jobs in the queue varies over time, an eligible job queue is used with the first 128 pending jobs. This value is the same as in RLScheduler and is also common practice in workload managers such as Slurm. The agent considers the jobs in the eligible job queue and their three corresponding attributes, composing the Job Observation, an array of size 128×3 **(1a)**. Simultaneously it obtains information about the nodes through the Node Observation array. It has as many rows as nodes in the data centre and three columns, one for each node attribute **(1b)**.

Next, the agent prepares the observation by merging the Job Observation and the Node Observation attributes of each combination of job and node, and adds an extra value called *Can Be Scheduled* **(2)**, which is defined as $c_{j,n} = n_j \leq fp_n$ resulting in true if the node n has room for the job j and false otherwise. This combined observation is a matrix with $128 \times NumNodes$ rows and $JobAttributes + NodeAttributes + 1$. Here, the rows represent all the possible pairings of jobs and nodes, and the columns are the total number of attributes

that define each of these pairs, which in this instance it is equal to seven. The fact that the Node and Job Observations are combined serves the purposes of presenting the agent all the possible pairings of nodes and jobs, and allow it to make a decision with sufficient information.

The next step is to let the agent select from the observation the job-node pair to be scheduled. This decision is reached with the aid of the actor network of the agent **(3a)**, which has seven inputs, one for each attribute in the observation. Then it has three fully-connected hidden layers of 32, 16 and 8 neurons each, with ReLU as their activation function. Finally, the output layer is of size 1, as purpose of this network is to provide a single score value for each jobs-node pairs. The actor is fed the whole observation matrix, therefore, the output is also not a single score but a column vector of $128 \times NumNodes$ scores. Then, a mask is applied to the score vector to remove the values corresponding to padding jobs added to complete the observation, or those that request more processors than those free in the node **(4)**. This way, any job that the agent may choose is assured to be able to be scheduled without waiting for resources to get free.

Next, a softmax function is applied to the masked vector, transforming the scores into a probability distribution in which the sum of all elements is 1 **(5)**. With these probabilities, an action is selected that will indicate the job and node to be scheduled next, favouring those with higher score. In production this step is skipped and the job-node pair with the highest score is chosen. The job-node pair is an integer $a_i \in [0,\ 128 \times NumNodes - 1]$, the index of the job is calculated as $\lfloor \frac{i}{128} \rfloor$ and the node as $i \bmod NumNodes$ **(6)**.

This action is passed on to the simulator that executes the corresponding scheduling operation. After the simulator advances the time to the next event, which results in a new state of the environment, a reward **(7)** will be obtained based on the chosen objective, together with a new observation representing the new state of the environment. These are used by the critic network in the agent **(3b)** to evaluate the performance of the actor network. It guides the training process of both networks toward a state where the agent consistently schedules the jobs to the right computing resources, such that the objective is satisfied. The goal of the critic network is to predict the reward that a set of jobs would produce with the given objective.

Three reward functions have been implemented, each corresponding to a different scheduling metric to be minimised. If e_i and w_i are the execution and wait times of job i, the metrics are

- *Slowdown (SLD)*: is the average slowdown, defined as $\frac{w_i + e_i}{e_i}$, for all the jobs. This metric can give very high values when jobs are short.
- *Average bounded slowdown (BSLD)*: variation of the slowdown that is less sensitive to very short execution times. The bounded slowdown of a job is defined as $max((w_i + e_i)/max(e_i, 10), 1)$.
- *Average waiting time (AVGW)*: simply averages w_i of all jobs.

It is worth noting that all the objectives are related, since they strive to reduce the delay in the execution of the jobs. However, the average the bounded slowdown metric is better suited to agent training than the other two. First,

it conveys more information than the average waiting time because it includes the execution time of the jobs, and second, it is more stable than the average slowdown, since it eludes giving very high values due to short execution times.

3.3 Size Reduction Through Clustering

A drawback of this agent is the large size of the input. For instance, in a data centre with 16 nodes, the total number of elements of both observations would be $128 \times 16 \times 7 = 14336$. Doubling the number of nodes in the data centre results in 28672 elements, which has a clear impact in the performance and the scalability. Since the size of the queue is fixed to 128, reducing the size of the observation can only be done by limiting the size of the node observation. This section describes how this can be accomplished with clustering techniques.

In a data centre, it is common that many nodes have a similar situation, either due to their equivalent architectural properties or similar load. Then, it is not necessary to identify exactly which node is going to receive a job, and it suffices to indicate what kind of node is the target. Taking this into account, the n nodes of the data centre can be grouped in k *clusters* of similar attributes using the *k-means* algorithm [5]. The attributes of each cluster are calculated as their mean value for all the nodes in each one. This is applied to the node observation **(1b)** before it is merged to the job observation (Fig. 1), which now carries job-cluster pairs instead of job-node pairs. Like before an attribute c is calculated, indicating if the job fits in at least one node in the cluster.

The final step after the job-cluster selection has been made is to choose a specific node for scheduling the job, which is done by simply finding the first node of the cluster that can execute the job. This selection does not need any further considerations, as the assumption is that nodes in the same cluster are similar enough. By grouping the nodes in a fixed number of clusters, the size of the node observation becomes constant. Thus, it is possible to increase the number of nodes in the platform without complicating the agent.

4 Evaluation

The proposed agent is evaluated through four instances. The *SqSLD agent*, *SqBSDL agent* and *SqAVGW agent* aim to minimise the slowdown, average bounded slowdown and waiting time, respectively. The *ClBSLD agent* uses clustering of the compute resources to minimize the average bounded slowdown, although any of the other two objectives could have been employed.

The target system is a heterogeneous data centre with 20 nodes. Each can have between 4 and 64 processors, running at 2, 2.5, 3 or 3.5 GHz. The workload used is generated from models defined in the Parallel Workloads Archive, *Lublin, 1999/2003*, commonly used in HPC [3,8]. This workload is composed of 10 000 jobs with varying required processors and execution times.

Also, a set of heuristic algorithms were used for comparison. They are able to select jobs and the nodes to execute them. These result by combining two algorithms, one to choose the job and another for the resource. These are summarised in Table 2, then algorithm xy combines job selection x with node selection y.

The hyper-parameters used to control the training process of the agents are mostly the same as in RLScheduler. The most relevant ones are the learning rate α, with values of 0.0003 and 0.001 for the actor and the critic networks, respectively, and gamma γ is equal to 0.99.

Table 2. Heuristic job and node selection algorithms.

Name	Symbol	Description
Job Selection		
Random	r	Random job from the job queue is selected
First	f	Job with lowest submit time is selected
Shortest	s	Job with lowest requested run time is selected
Smallest	l	Chooses job with lowest requested number of processors
Node Selection		
Random	r	Random node is selected
Biggest node	b	Node with highest number of processors is selected
Fastest node	f	Node with highest frequency is selected

To explore the training phase, each instance of the agent is subjected to 100 epochs and the evolution of the process is observed to ensure that it converges. To lighten this process, the workload trace is not used in its full length. One training epoch consists of 20 *trajectories*, which are sets of 256 consecutive jobs, taken at a random time from the original trace. The experimental results show that 100 epochs are more that enough because convergence was observed after 60 epochs, since the behaviour did not improve in the following epochs.

Once the training phase concludes, the inference stage is evaluated. The trained agents must schedule trajectories of 1024 jobs extracted from the same workload. Note that the presented results consider 20 repetitions to avoid obtaining wrong conclusions due to outliers. The scheduling results are evaluated by comparing the behaviour of the trained instances to that of the heuristic algorithms. These are shown in graphs where the horizontal axis represents the values of the metrics used, and the vertical axis shows the different schedulers, sorted by the median. To avoid clutter, only the results for the best heuristic algorithms are shown. The graphs combine box-and-whisker and violin representations of the results. The box shows the 25 and 75 percentile, the line in the box indicates the median, and the whiskers represent extreme values. The violin plots show result distribution, where fatter parts indicate a higher data density.

Figures 2, 3 and 4 show the promising results of the four agents. In general, the results prove that intelligent schedulers can perform better than the state-of-the-art algorithms in a heterogeneous data centre, at least for the objectives considered in this article.

Note how the algorithms that select the random or first jobs, the first six in the graphs, give very bad results. In some cases, more than tripling the results

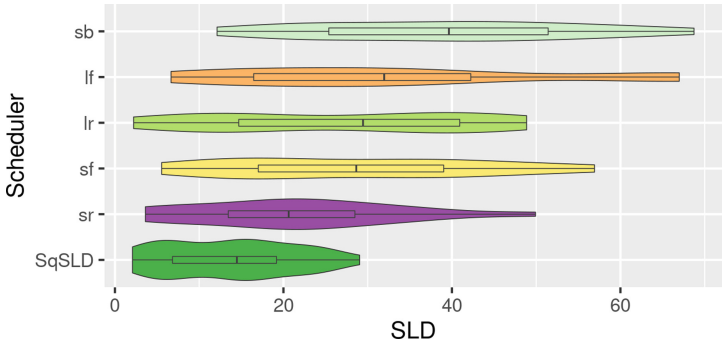

Fig. 2. Average slowdown results for heuristic algorithms and SqSLD Agent.

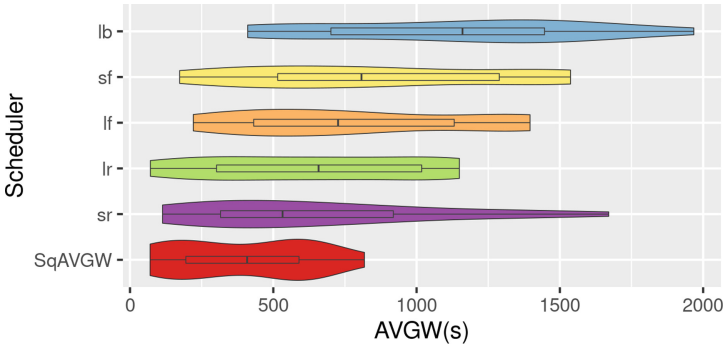

Fig. 3. Average waiting time results for heuristic algorithms and SqAVGW Agent.

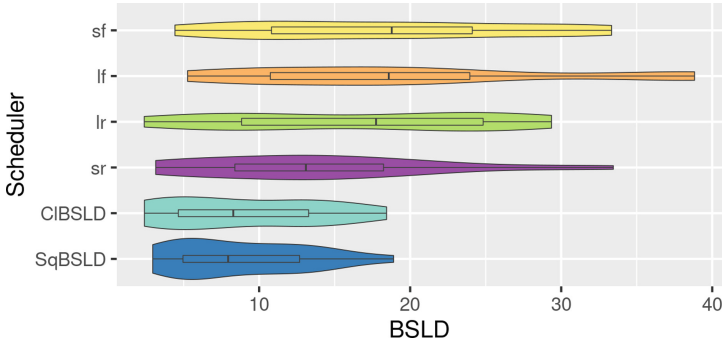

Fig. 4. Average bounded slowdown results for heuristic algorithms, SqBSLD and ClB-SLD Agents.

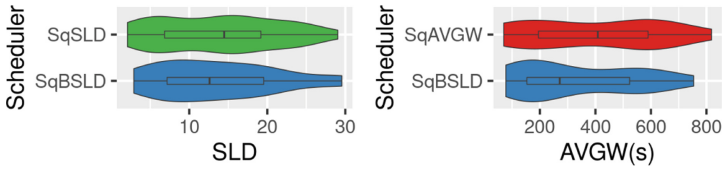

Fig. 5. Agent performance with metrics different of the one used for training.

of the corresponding agent. The algorithms that choose the shortest or smallest jobs perform better, especially with the random or fastest resource selections.

Considering only the heuristic algorithms, the graphs show the importance of choosing the right node for a job in the context of a heterogeneous data centre. Note that only the algorithms that choose the shortest or smallest jobs first appear in the graphs. The rest had significantly worse results and were excluded to avoid clutter. It can be seen that *sr* has always the lowest median, followed by *sf* or *lr*, depending on the metric. As for node selection policy, the best results are obtained by either random or fastest. It is noticeable that *lr* presents the lowest values in all three metrics.

However, all these algorithms are always bested by the intelligent agents. Indeed, the graphs show that the median is always lower than that of the best heuristic algorithm *sr*, also the minimum values of the agents are similar to the *lr*. But in all cases they have lower variance than any of the heuristics, ensuring that good results are given in a more consistent manner.

An improvement was proposed where the complexity of the agent was reduced by incorporating a clustering algorithm to group the nodes of the data centre. As this evaluation aims only to establish the cost-benefit relation of adding the clustering vs. reducing the complexity of the agent, only one objective has been tested, the average bounded slowdown. The 20 nodes of the system were grouped into 10 clusters, so the complexity of the DNN was reduced in half.

The results of the ClBSLD Agent (Fig. 4) are comparable to those of the SqBSDL Agent. The minimum result given by the ClBSLD is smaller than the SqBSDL, but since it has higher variance, the median ends up being slightly higher. At any rate, the experiment proves that the clustering method can be applied in cases where the combined observation has become too large, and many of the nodes have the same or very similar characteristics.

It is interesting to observe the results of the agents with metrics different to the ones used for training. To this aim, the SqBSDL Agent instance, trained to minimise the bounded slowdown, was selected and tested with the other metrics, average slowdown and waiting time. The results are shown in Fig. 5, compared to the results of the SqSLD and SqAVGW Agents. In both cases the results of the agents are roughly similar. Although the median values of SqBSDL are lower than those of the other two, the best case results are always obtained by the other instances. This is explained by the fact that bounded slowdown is better suited to agent training, and therefore, it is able to give a better scheduling.

The above evaluation proves that an intelligent agent is able to learn how to take scheduling actions and obtain better results than classic scheduling algorithms. And this can be done not for a single goal but for different ones, only constrained by the capabilities of the simulator in which it is working. All this suggests that using a machine learning agent to schedule a real data centre is an idea worth considering. Provided that it is possible to obtain a trace of the jobs typically executed in the system to perform the training of the agent.

5 Conclusions

The fact that data centres are more and more heterogeneous, combined with the variety of the applications and their requirements, complicates scheduling significantly. With homogeneous clusters, heuristic algorithms are used to schedule jobs, but in heterogeneous ones it is crucial to decide also to which compute resource they scheduled. This problem is no longer possible to solve with such algorithms and there has been advances in employing machine learning instead.

This article presents a first approach to solving the scheduling problem in heterogeneous clusters with deep reinforcement learning. To this aim, it was necessary to redefine the observation space of the agent, allowing it to perceive more data from the environment. As well as to broaden the action space to accommodate the fact that not only jobs but also nodes had to be selected.

Also, two different agents were developed capable of successfully processing the state of a small heterogeneous data centre and learning to choose adequate scheduling actions. The second agent is a refinement of the first that through the use of clustering techniques is capable of giving the similar performance using a fraction of the memory requirements. The successful training of the agents was possible thanks to the development of a simulation infrastructure with a simplistic model of a heterogeneous data centre, that can simulate nodes with a different number of processors and frequencies.

The evaluation included in this article suggests that it is possible to replace heuristic schedulers with ones that leverage machine learning techniques. The experiments show that the behaviour of the machine learning agent gives very promising results, compared to well known heuristic algorithms.

Next developments could see larger clusters simulated with more detail, in which contention could be modeled, like that appearing in memory or network access. Furthermore, the set of objectives to optimise by the scheduler could be increased by considering energy related metrics.

Acknowledgment. This work has been supported by the Spanish Science and Technology Commission under contract PID2019-105660RB-C22 and the European HiPEAC Network of Excellence.

References

1. Bosque, J.L., Perez, L.P.: Theoretical scalability analysis for heterogeneous clusters. In: 4th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago, USA, pp. 285–292. IEEE Computer Society (2004)

2. Carastan-Santos, D., De Camargo, R.Y.: Obtaining dynamic scheduling policies with simulation and machine learning. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–13 (2017)
3. Feitelson, D.G., Tsafrir, D., Krakov, D.: Experience with using the parallel workloads archive. J. Parallel Distrib. Comput. **74**(10), 2967–2982 (2014)
4. García-Saiz, D., Zorrilla, M.E., Bosque, J.L.: A clustering-based knowledge discovery process for data Centre infrastructure management. J. Supercomput. **73**(1), 215–226 (2017)
5. Hartigan, J.A., Wong, M.A.: Algorithm AS 136: a K-means clustering algorithm. J. Roy. Stat. Soc. ser. C **28**(1), 100–108 (1979)
6. Herrera, A., Ibáñez, M., Stafford, E., Bosque, J.: A simulator for intelligent workload managers in heterogeneous clusters. In: 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp. 196–205 (2021)
7. Leonenkov, S., Zhumatiy, S.: Introducing new backfill-based scheduler for SLURM resource manager. In: Procedia Computer Science, 4th International Young Scientist Conference on Computational Science, vol. 66, pp. 661–669 (2015)
8. Lublin, U., Feitelson, D.G.: The workload on parallel supercomputers: modeling the characteristics of rigid jobs. J. Parallel Distrib. Comput. **63**(11), 1105–1122 (2003)
9. 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 (2016)
10. Mao, H., Schwarzkopf, M., Venkatakrishnan, S.B., Meng, Z., Alizadeh, M.: Learning scheduling algorithms for data processing clusters. In: Proceedings of the ACM Special Interest Group on Data Communication, p. 270–288. SIGCOMM 2019 (2019)
11. Pearl, J.: Heuristics: Intelligent Search Strategies for Computer Problem Solving. Addison-Wesley Longman Publishing Co., Inc, Boston (1984)
12. Pinedo, M.: Scheduling, vol. 29. Springer, Berlin (2012)
13. Stafford, E., Bosque, J.L.: Improving utilization of heterogeneous clusters. J. Supercomput. **76**(11), 8787–8800 (2020). https://doi.org/10.1007/s11227-020-03175-4
14. Stafford, E., Bosque, J.L.: Performance and energy task migration model for heterogeneous clusters. J. Supercomput. **77**(9), 10053–10064 (2021). https://doi.org/10.1007/s11227-021-03663-1
15. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT press, Cambridge (2018)
16. Tang, W., Lan, Z., Desai, N., Buettner, D.: Fault-aware, utility-based job scheduling on blue, gene/p systems. In: IEEE International Conference on Cluster Computing and Workshops, pp. 1–10 (2009)
17. Vazirani, V.V.: Approximation Algorithms. Springer Science & Business Media, Berlin (2013)
18. Yoo, A.B., Jette, M.A., Grondona, M.: SLURM: simple Linux utility for resource management. In: Feitelson, D., Rudolph, L., Schwiegelshohn, U. (eds.) JSSPP 2003. LNCS, vol. 2862, pp. 44–60. Springer, Heidelberg (2003). https://doi.org/10.1007/10968987_3
19. Zhang, D., Dai, D., He, Y., Bao, F.S., Xie, B.: RLScheduler: an automated HPC batch job scheduler using reinforcement learning. In: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15. IEEE (2020)