A Simulator for Intelligent Workload Managers in Heterogeneous Clusters

Adrián Herrera*, Mario Ibáñez†, Esteban Stafford‡ and Jose Luis Bosque§
Department of Computer Science and Electronics, University of Cantabria, Spain
Email: *adrian.herrera@alumnos.unican.es, †mario.ibanez@alumnos.unican.es,
‡esteban.stafford@unican.es, §joseluis.bosque@unican.es

***Abstract*—Modern High Performance Computing (HPC) clusters often comprise a huge amount of computing resources of different capabilities, making them *heterogeneous* and difficult to manage. In addition, they must deal with a wide range of applications with different requirements. All this poses a great challenge to the *workload managers* that assign applications to resources. There are many new proposals to overcome this challenge, including some that employ *Deep Reinforcement Learning (DRL)* techniques. This paper proposes a novel simulation framework for the study of workload managers, that has been conceived to foster the study of workload managers based on DRL techniques. Its main features include the simulation of heterogeneous clusters based on multicore architectures, taking into account the contention in shared memory access and the energy consumption. A validation of the accuracy and performance of the simulator was made, compared with a real environment based on Slurm. This shows good accuracy of the results, with a relative error below 5% in makespan and 10% in energy consumption, and speedups up to 200.**

***Index Terms*—Resource Management, Reinforced Learning, Scheduling Simulation, Heterogeneous Systems.**

I. Introduction

Efficiently managing the workload of High Performance Computing (HPC) clusters has always been an arduous challenge. This task, carried out by a *workload manager*, requires a complex decision making process which allocates available resources to user jobs, honouring job requirements while optimizing for certain performance and energy consumption objectives. To further complicate this, there is a scale problem; in the last decades clusters have shown a high increase in number of resources, designed to meet the increasing demand from users. Previous literature offers a rich set of scheduling algorithms to address this problem, like [27].

Workload management is an instance of the *job shop scheduling* problem, which has been proved to be NP-hard for large amounts of resources [10]. Fortunately, it appears to be well suited to the application of machine learning techniques, such as neural networks [15], [32]. These usually require a training phase that exposes the neural network to a large number of realisations of a given experiment, called *episodes*, allowing it to learn to take the best decision. In the case of workload management, each of these episodes consists of the submission of a set of user jobs to the cluster. If a neural network needs tens of episodes, the time and energy cost of training becomes prohibitive. One way to overcome this cost is taking advantage of simulators.

The simulator required to analyse the performance of workload managers must necessarily be able to simulate whole computer clusters. As with any simulator there is a tradeoff to be met, balancing the need for simulation speed and the accuracy of the results. Since the training process of a neural network is so time consuming, the simulator used must be extremely fast. However, it must not oversimplify the architectural model of the cluster, or the neural network will learn to operate an ideal environment, and will not make the correct decisions in a real cluster.

This article proposes IRMaSim, a cluster simulation framework with the following features. It organises the computing resources in a hierarchical manner, permitting the correct representation of multi-core and multi-processor computers. This allows that each element can have performance and energy parameters, like clock frequency, memory or energy consumption, permitting the representation of heterogeneous clusters. It also enables the implementation of schedulers that address optimisation of one or more different objectives. Furthermore, the hierarchy naturally groups elements around shared resources and the simulator can model contention, like it occurs with cores sharing the memory channels of the processor they are in.

As with any simulator, IRMaSim must balance two requirements, speed and accuracy. The availability of an extremely fast simulator is key to allow researchers to make the large amount of simulations necessary to tune parameters or simulate a number of different scenarios. All this is conditioned to the accuracy of the simulations, as the behaviour of a simulated workload manager must be equivalent to its real execution. Therefore this article presents a validation of the simulator against a real cluster managed with Slurm, showing that the relative error in performance and energy consumption are at most 5% and 10%, respectively, while boasting speedups up to 200.

Being able to simulate heterogeneous clusters is important because it allows representing a common case nowadays, where clusters have a wide collection of nodes with different computing capabilities [4], [5], [12]. In addition, the variety of the applications has grown, ranging from classic number-crunching scientific programs [3] to memory-hungry big-data applications [21]. In this situation, workload managers must be able to assign applications to the nodes most suited to their characteristics. Then, a simulator that does not take

978-1-7281-9586-5/21/$31.00 ©2021 IEEE
DOI 10.1109/CCGrid51090.2021.00029

into account these facts will not be a good tool to test heterogeneity-aware schedulers [25].

The way in which computers are designed presents compute resources sharing a set of subsystems, like cores or processors accessing main memory or compute nodes sharing network or storage. This situations eventually lead to contention in the access to the shared resources, causing a significant degradation in the performance of the applications. This fact must be acknowledged by a workload manager trying optimise the assignment of user jobs to cluster resources. Therefore a simulator of this kind must model contention correctly in order to present a more realistic behaviour of modern clusters.

Traditionally, workload managers are aimed at optimising performance metrics, like makespan or throughput. However, energy consumption is becoming a major concern [16], [19] inspiring the appearance of energy-saving schedulers. But since the objectives of performance and energy efficiency are usually opposed, some efforts have been made in the study of multi-objective schedulers. This reinforces the fact that a simulator must take energy consumption into account to allow scheduler designers to evaluate these new objectives.

Finally, given the rise of schedulers taking advantage of machine learning techniques [15], [32], IRMaSim offers an API suitable for the design of this kind of schedulers. For instance, it can allow the utilisation of *Deep Reinforcement Learning (DRL)* [26] to select the best policy to satisfy a specific objective. This is accomplished by an *agent* making an observation of an *environment*, based on which it makes a decision that alters the environment, and receives a *reward* that modifies the way the agent makes the next decision.

To the authors knowledge this is the first simulator that offers all these features to the workload manager designers.

The main contributions of this paper are the following:

- Extending the Batsim simulator to model multicore architectures, taking into account memory contention and energy consumption.
- Providing an easily extensible framework to research DRL techniques in the field of workload management.
- Presenting an experimental validation of IRMaSim based on a traces extracted from a real cluster.

The remainder of the paper is organised as follows. Section II motivates the need for the IRMaSim simulator. Section III explains the models that were developed to simulate HPC clusters, while the machine learning support of the simulator is presented in Section IV. Section V delves into how the models and machine learning support were implemented. Then, Section VI presents some decisions regarding the validation of IRMaSim, which is undertaken in Section VII. This is followed by an account of the related work found in the literature, in Section VIII, and some concluding thoughts in Section IX.

II. Motivation

IRMaSim is built upon *Batsim* [8], a simulator for batch scheduler analysis. It was selected because it can model heterogeneous cores, for its easy extensibility and the possibility of integration with other libraries. However, it models clusters through a set of independent computing resources, like cores, and their associated memory [8]. This approach is very far from current multicore architectures which usually define a hierarchical structure that forces cores to share access to given resources, like memory, or the energy consumption of common parts of the processor, like the last level cache or the memory controllers.

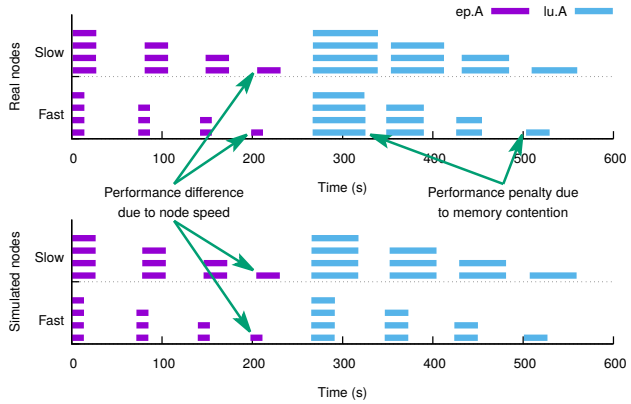

Fig. 1. Real execution and simulation on two nodes of different speed.

These discrepancies have a significant impact in the accuracy of Batsim. Figure 1 shows the results of a very simple experiment, with two quad-core nodes executing a set of jobs. The figure compares the behaviour of the execution in real hardware to the corresponding Batsim simulation. The nodes have different computing speed (Fast, Slow), and the jobs are 10 executions of ep.A and lu.A, two benchmarks of the well known suite NPB [2]. The horizontal axis represents time in seconds, while the vertical axis represents the nodes. Each horizontal line in the graphs shows when the task was scheduled and its execution time.

Looking at the executions of the ep.A benchmark, Batsim correctly models the performance difference of both nodes, even when there is more than one task running simultaneously in each node. However, the lu.A benchmark is memory-bound, which has an impact in the execution time when more than one job is scheduled to the same node. Since the four cores in real nodes have to share the memory access bandwidth, applications suffer a performance penalty compared to running alone, up to four times. To make matters worse, the penalty is not consistent in both nodes, as it is worse in the Fast node. With longer simulations, and with larger clusters, these errors will accumulate over time and lead to highly inaccurate results.

In summary, Batsim does not adapt well to the current trends in clusters, as its results differ both in terms of execution time and energy consumption. And consequently, the simulation of a scheduler will not show the same behaviour as a real one.

III. Architectural Modeling

These shortcomings of Batsim are addressed in IRMaSim through a more detailed modeling of the architecture of the cluster. To this aim, two major improvements were made and

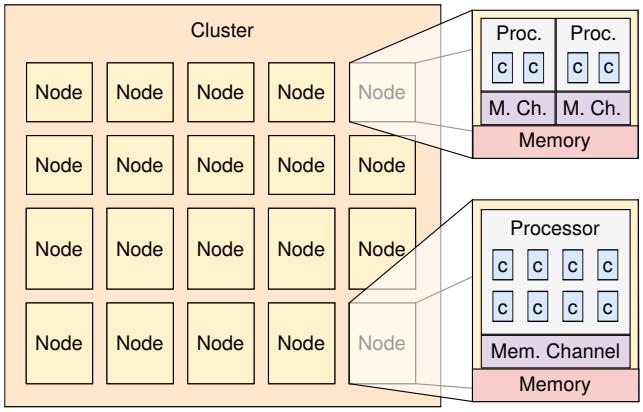

Fig. 2. Heterogeneous platform example.

described in the following sections. First in the definition of the platforms, that represent the architecture of the cluster, and second, in the execution models, that predict the performance and energy requirements of the running tasks.

A. Platforms and workloads

In the simulator, the cluster with its computational resources is represented by a *platform*. The improvement IRMaSim makes is that these are organised in a hierarchical manner to model current multicore architectures. Thus allowing the simulation of the different computational resources, while naturally including the concept of shared resources.

Figure 2 shows an example of a simple platform that models a heterogeneous cluster. In turn each node includes one or more processors and a main memory, and each processor contains one or more cores that share a memory channel. In more detail, the components modelled by the simulator are:

- *Platform* is the root element of the system.
- *Nodes* group several processors that share a memory resource. Equivalent to a server with its sockets and memory modules.
- *Processors* can have one or more cores that access the memory of the node through shared memory channels.
- *Cores* are the minimal computational unit in the system. It has individual performance and power consumption.
- *Memory and channels*: every node has an amount of memory, and each processor in the node has channels with a given memory bandwidth, that are shared by the cores.

This allows a scheduler to have the knowledge of how the computing resources of the cluster are related and therefore make scheduling decisions accordingly. For instance, it might attempt to save energy by sharing nodes among tasks or improve performance by giving whole nodes to each task.

The workload to be simulated is a sequence of jobs that are read from a trace file. Each job is defined with a number of parameters, of which the following are worth mentioning:

- *subtime*: is the time at which the job is submitted to the queue.
- *res*: is the number of tasks in the job. Each task is allocated to one core.
- *cpu*: real amount of instructions that will be executed by the job. It is important to note that the workload manager does not know the real running time of each job when making decisions, it is only seen by the simulator.
- *ipc*: is the average number of instructions executed per cycle.
- *mem_vol*: is the amount data that is sent and received from the main memory.

From the above, *cpu*, *ipc* and *mem_vol* are easily obtainable through a profiling tool like Likwid [23].

B. Execution and energy models

The objective of the execution model is to correctly predict the execution time of a task, taking into account the heterogeneity of the cluster and the performance impact of memory sharing. This is done by first calculating the execution time of the task in a specific node and then applying a slowdown factor in the case of memory contention:

$$T_{exe} = \frac{I}{IPC \cdot f} S_m \tag{1}$$

where I and IPC are the number of instructions and instructions per cycle of a task, respectively; f is the clock frequency of the node; and S_m is the memory slowdown. This last factor is a value between 0 and 1, that will be 1 in the absence of contention and lower values as the contention increases.

To model the impact of memory contention the methodology employed is based on an empirical analysis followed by a regression study. The first phase consists in determining how the performance of a task is degraded depending on its memory access rate and that of other tasks in the same node, and thus, sharing the memory bandwidth. To model the behaviour of tasks, a synthetic benchmark has been used [29]. It has a sustained memory access rate throughout its execution and can be set to different values. This behaviour is suitable to model scientific tasks, which have an iterative nature and the ratio of computing operations to memory accesses is fairly constant.

Figure 3 shows the memory slowdown of a task of interest running together with another three tasks in a four core processor. The X axis represents the memory access rate of the task of interest R_i if it were running alone. Similarly, the Y axis indicates the sum of the memory access rate R_o^j of the other three tasks ($j = \{1, 2, 3\}$) in the node if each of them were executing in isolation. The figure shows a plateau with memory slowdown equal to 1 where the access rates of the tasks is less than the memory bandwidth of the node. As the memory requirements of the tasks increase, the memory slowdown decreases to a minimum. Its important to note that the decrease, especially in slices with constant X, has a sigmoid shape. In the figure the minimum value is $0.25 = \frac{1}{4}$ and other experiments with n number of tasks have confirmed

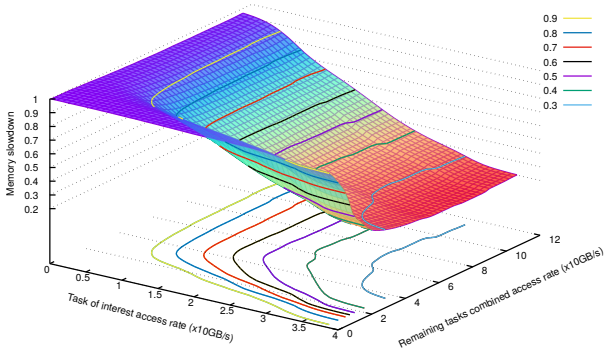

Fig. 3. Memory slowdown of a task of interest running simultaneously to three other tasks.

that in general the minimum slowdown is $\frac{1}{n}$. It is noticeable that the performance penalty of the task of interest depends not only on its access rate, but also that of the all the tasks sharing the node $R_T = R_i + \sum_{j=1}^{n} R_o^j$. Therefore the model must depend on these three values.

With these observations, the regression analysis was attempted with a number of functions that resembled the sigmoid shape. The one found to be most adequate was a linear piecewise function (Eq. 2), composed by two horizontal half lines joined by an oblique segment. The half lines represent the maximum and minimum slowdown values, while oblique segment represents the interval in which the performance degrades between the previous values.

$$pwl(R_i, R_T, n) = \begin{cases} a & R_T < c \\ b(R_T - c) + a & c \geq R_T > d(R_i, n) + bc - a \\ d(R_i, n) & R_T \geq d(R_i, n) + bc - a \end{cases} \quad (2)$$

In Equation 2 the value of a is one, meaning there is no performance penalty at low memory access rates. The value of c represents the point at which degradation commences, and b the rate at which the performance decreases. The latter values are constant, depending solely on the compute node. Finally, d is the minimum memory slowdown, but its value strongly depends on the number of tasks sharing the memory (n) and the access rate of the task of interest (R_i). Therefore a second regression analysis was applied on it, yielding the following expression:

$$d(R_i, n) = \frac{ss\left(\frac{R_i - (da - n)db}{dc - n \cdot dd}\right) n + 1}{1 + n} \quad (3)$$

where da, db, dc and dd are constant values depending on the compute node. These are fixed by the regression algorithm based on the data obtained in the empirical analysis. Finally, $ss(x)$ is the 5^{th} order smooth step function (Eq. 4).

$$ss(x) = \begin{cases} 0 & x < 0 \\ 6x^5 + 15x^4 - 10x^3 & 0 \geq x > 1 \\ 1 & x \geq 1 \end{cases} \quad (4)$$

To complement the performance model, another was developed to estimate the energy consumption of a cluster. In a multicore architecture the power consumption of a processor is composed by two values. First, the consumption of the cores themselves. And second, a fixed amount corresponding to common circuitry units, like memory controllers or last level cache. Therefore the first value grows linearly with the number of active cores, while the second is constant for each processor, corresponding to the following model:

$$P(n) = pa \cdot n + pb \quad (5)$$

where pb is the power consumption of the common units of the processor, and pa is the increase of power consumption for each additional active core. This behaviour can not be modeled by Batsim because each core is completely independent from each other.

The values for the model can be obtained through an regression analysis. First choosing a compute-intensive benchmark, and performing a set of executions on a multi-core machine, varying the number of concurrent tasks from one to the number of cores. The above equation can be fitted to the results of this experiment, yielding the values of pa and pb.

IV. Deep Reinforcement Learning Support

The problem of workload management in modern clusters is one that requires the treatment of large amounts of data, both coming from the tasks and from the state of the cluster itself. This leads to a challenging decision making problem where artificial intelligence techniques may be applied. DRL has been successfully applied to the problem of workload management in the past [15].

In order to further test these ideas, a major target of IRMaSim is allowing the development of DRL scheduling algorithms, like the one depicted in Figure 4. The interaction between the *agent* and the *environment* is structured in *steps*. Each starting with the agent receiving the *state* of the environment plus a *reward*, and followed by the agent issuing an *action* over the environment. Every action alters the environment, changing its state and producing the reward of the following step [26].

A. Agents

The entity in charge of learning and taking scheduling decisions is known as the *Agent*. These are in fact a combination of two separate sub-decisions: job selection and resources allocation. The learning behaviour is based on the *gradient-descent* algorithm.

The agent receives scheduling events, that can be a new job that arrives when the queue is empty, or that some resource has been released in the platform. Both of these trigger a

simulation cycle in which the agent carries out the following steps:

1) The agent makes an observation of the environment and receives a new reward value, based on the impact of the previous actions.
2) The observation is fed to the *inner model*, which may be based on an artificial neural network or, in simpler cases, a static mapping between observations and actions. And this selects an *action*.
3) The agent learns through an adjustment of the weights of the neural network, based on a *loss function* that takes into account the reward. This brings the agent closer to optimising the selected objective.
4) The action, which is a scheduling policy, is applied to the jobs in the queue. Actions produce alterations in the environment, as a consequence of allocating resources to jobs. These alterations are not directly observable by the agent, they will be evaluated in the next step.

IRMaSim allows to easily implement a variety of agents, and as an example, it includes two common solutions: *reinforce* and *actor-critic*. Both agents are depicted in Figure 5, where reinforce is the upper portion denoted as actor network. The input layer receives the observation, which is then forwarded through the hidden layers, before reaching the final output layer. The latter is where the output is generated. Every layer is *dense*, meaning that each neuron in a layer receives input from all the neurons present in the previous layer.

The actor-critic [14] agent extends the actor network with a similar one, shown in the bottom part of Figure 5. The critic network is meant to learn a value function that the actor network can use to update its parameters to improve the performance. Both agents are implemented using gradient descent, that compared to other methods increases the probability of convergence.

IRMaSim allows choosing from a number of objectives to optimise. *Makespan* reduces the time from the arrival of the first job until the completion of the last job. *Energy consumption* minimises the total amount of energy consumed for the workload. And *Energy efficiency* minimises the energy delay product of the workload's execution. For each of them a loss function and reward function has been defined and can be seen in Table I.

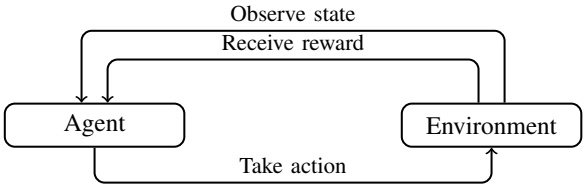

Fig. 4. Reinforcement learning loop

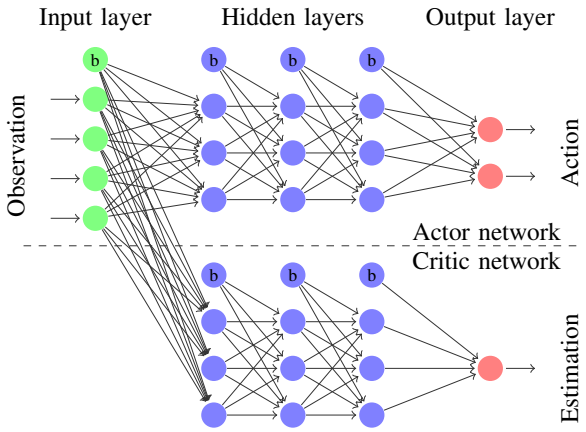

Fig. 5. Inner model of a Reinforce agent with actor-critic network extension.

TABLE I
OBJECTIVE AND REWARD FUNCTIONS

C_j: time between arrival and completion of a task.
C_r: computing capability of resource.
J: number of jobs in the workload.
R: number of computing active resources.
P_r: total potency consume by a resource during simulation

Objective	Function	Reward
Makespan	$\min \sum_j^J C_j$	$\sum_r^R C_r$
Energy Consumption	$\min \sum_r^R P_r \sum_j^J C_j$	$-\sum_r^R P_e \sum_j^J C_j$
Energy Efficiency	$\min \sum_r^R P_r \sum_j^J C_j^2$	$-\sum_r^R P_e \sum_j^J C_j^2$

B. Environment

In IRMaSim the *environment* comprises the platform, representing the resources of the cluster, and the workload, enumerating the tasks that will be executed (Section III-A). In DRL terms the agent interacts with the environment through an *observation space* and an *action space*.

1) Observation Space: The observation space must convey the state of the nodes in the platform, as well as the jobs in the queue. However, real clusters involve thousands of computing resources, each with several parameters, such as compute load, power consumption or memory usage. Furthermore, the workload queue can have a high number of jobs, each with start times, number of instructions and memory requirements. Sending all this information to the agent is impractical, therefore a novel observation space has been implemented in IRMaSim. This tries to provide a compact and useful information summary:

1) For each node, the fraction of memory capacity available.
2) For each core, the current computing capability and power consumption fractions. Also the remaining workload of the task in execution, which this is calculated from the user-provided and current execution times.

TABLE II
CLASSIC POLICIES EXPOSED THROUGH THE ACTION SPACE

Job Selection Policies	
random	Any job scheduling
first	Job with earliest submit time
shortest	Job with shortest user requested time
smallest	Job with lowest user requested cores
low_mem	Job with lowest user requested memory
low_mem_bw	Job with lowest user requested memory bandwidth
Resource Selection Policies	
random	Any resource
high_gflops	Resource with highest current FLOPS
high_core	Resource with most available cores
high_mem	Resource associated to the node with most memory
high_mem_bw	Resource associated to the processor with most memory bandwidth
low_power	Resource with lowest current Watts

3) For each job parameter, *time*, *cores*, *memory* and *mem_vol*, five statistics are included: the minimum, Q1, median, Q3 and maximum quantities in the whole job queue. This gives the agent a holistic view of the queue state instead of accurately presenting specific part of it.
4) Finally, a variation ratio of the queue size with respect to the last observation.

Feature scaling is applied to each of these values, constraining them to the range $[0, 1]$, to equalise their weight in the decision process. Data is arranged in a 1-D vector and sent as input to the agent. Nevertheless, the size of the observation vector can still be too big. Therefore, IRMaSim offers three levels of detail. The *normal* level provides all data previously listed, with a size of $Nnodes + Nprocessors + Ncores * 3 + Nrestype * 5 + 1$ items. The *small* level skips the per-core information. And the *minimal* level provides only the job distribution and variation ratio.

2) Action space: The agent alters the environment through actions. The *Action Space* is the set of actions which can be selected by the agent. IRMaSim implements a discrete action space where several classic policies are presented to the agent to choose from. A policy consists of a pair of *job_selection_policy* and *resource_selection_policy*, which are shown in table II. A special *void action* is also incorporated in order for the agent to be able to stall; this is intended to address cases where the selection of any other action would result in a worse outcome. In total there are 37 policies, 36 combinations from classic policies plus the void action. Nevertheless, IRMaSim is designed to be easily extended in this sense. Any subset of policies might be specified by the user to adjust the action space size in their experiments.

V. IRMASIM DESIGN

As stated before, IRMaSim takes advantage of the functionality of Batsim and PyBatsim. A great effort has been made to isolate new functionality from these tools and keep them intact. Upon these two frameworks, IRMaSim builds the necessary components to simulate heterogeneous systems with resource sharing constraints and deep learning decision systems. An overview of the design in three layers is shown in Figure 6. The first being the simulation instance in Batsim. Second is the PyBatsim layer, that eases the communication to Batsim, as well as parsing the workload traces. The last layer is IRMaSim whose components are detailed next.

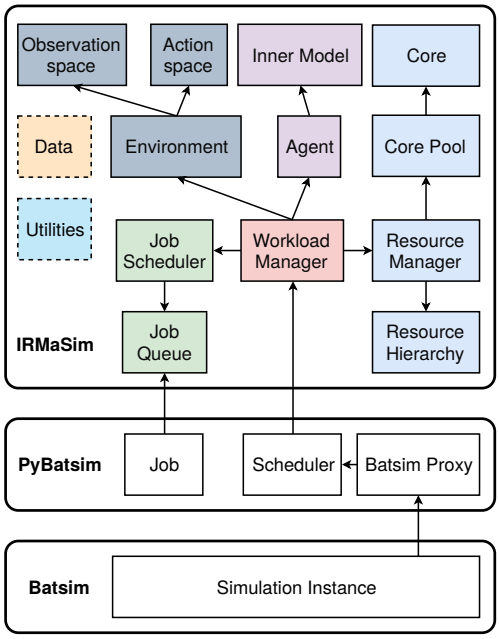

Fig. 6. Full component layout

The *Job Queue* receives jobs from PyBatsim and holds the jobs that are eligible for execution. The *Job Scheduler* selects from the Job Queue the next job that will be sent to the *Workload Manager*. For the selection any scheduling policy may be used from the ones listed in Table II.

The *Workload Manager* is the entry point for the decision system. It communicates with PyBatsim via events, which can be job submissions, job completions, resource allocations and releases. In every cycle, it will receive jobs from the Job Scheduler and through the Resource Manager resources will be allocated for them. Then, the mapping between jobs and resources is sent to PyBatsim. The *Resource Manager* selects resources available for new jobs using a selection policy from those listed in Table II. Once a resource is allocated to a job, the Resource Manager updates performance and energy models of the affected resources in the *Core Pool* and *Resource hierarchy* components. The latter provides relations between each type of resource, and allows for the *Resource Manager* to determine which cores share resources, like memory.

To support the implementation of DRL techniques, IRMaSim has been extended by adding event flow control between actions taken by the agent, information reported by Batsim and observations made in the Environment. These new components that have been already described in previous sections are the *Agent* with its *Inner-Model* and the *Environment*

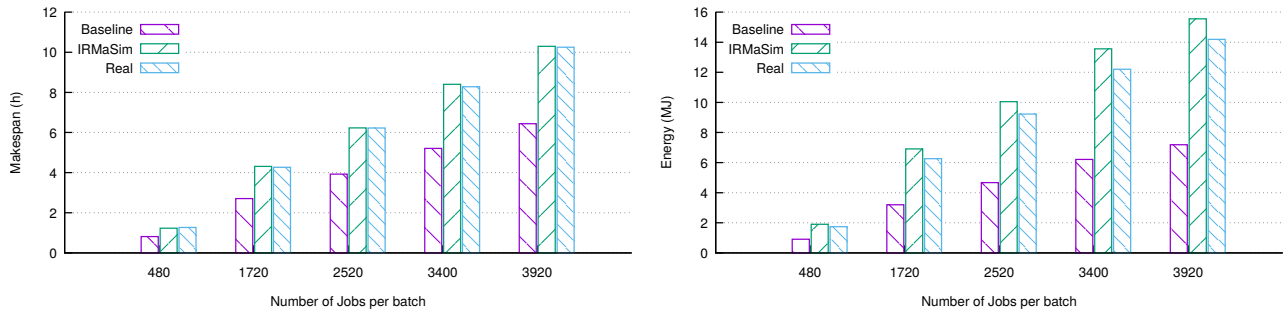

Fig. 7. Makespan and energy results in a homogeneous cluster.

TABLE III
MODEL PARAMETERS FOR COMPUTE NODES.

Freq	b	c	da	db	dc	dd	pa	pb
3.4 GHz	$-1.85 \cdot 10^{-5}$	32000	1.75	3500	45000	3000	6.14	5.59
2.5 GHz							3.51	3.20
1.7 GHz							2.31	1.59

that includes the *Action Space* and the *Observation Space*. Two python libraries have been used to facilitate the implementation of the machine-learning features, PyTorch [18] to create the agents and inner-models, and the OpenAI's Gym library to provide a standardized *environment* and action/observation *space* definitions [6].

VI. METHODOLOGY

For the purpose of validation, this article compares simulation results to the execution of a set of benchmarks on real hardware. The cluster used is composed by 15 compute nodes and one front-end node, each with a Intel Core i5-7500 Kaby Lake architecture with 4 cores and two memory channels with a combined bandwidth of 38.4GB/s. The memory configuration consists of two 4GB of DDR4-2400 modules. The nodes are connected to each other with GigaBit Ethernet. The cluster is based on CentOS 7.6 and the workload manager is Slurm 17.11. The parameters for the memory contention and energy models are listed in Table III.

The traces fed to the simulator are extracted from a set of batch executions of benchmarks selected from the NPB suite version 3.3.1 [2]. The list of benchmarks together with their properties, measured with Likwid [23], is shown in Table IV. Each batch is a burst of sequential jobs randomly selected from the previous list. Since the serial versions of the benchmarks are used, the concept of job and task are in some cases interchangeable. The jobs of each batch are submitted in the first minutes, so the queue is never empty until the end of the batch. For each experiment, there are five batches with growing number of jobs and therefore execution time.

As with any simulator, there is a need of compromise between the speed of the execution and the accuracy of the results. Therefore the following metrics have been considered in this validation. The makespan, defined as the time from the first job submission to the conclusion of the last task.

TABLE IV
EXECUTION METRICS OF BENCHMARKS.

Benchmark	Instructions	IPC	Mem. vol.
bt.C	$633.50 \cdot 10^{10}$	3.06	5.17TB
cg.C	$60.31 \cdot 10^{10}$	1.10	7.30TB
ep.C	$80.94 \cdot 10^{10}$	1.16	0.21TB
is.C	$6.43 \cdot 10^{10}$	0.95	0.07TB
lu.C	$370.25 \cdot 10^{10}$	2.20	8.70TB
sp.C	$356.10 \cdot 10^{10}$	2.74	9.60TB
ua.C	$312.57 \cdot 10^{10}$	2.30	4.32TB

The energy consumption is the total energy consumption of all the nodes during the makespan. The execution time of the simulator itself is also measured. To show the improvements of IRMaSim, experiments show results of a *baseline* model that represents the cluster as a pool of independent cores, not taking into account the multicore architecture.

VII. VALIDATION

The validation of the IRMaSim simulator is accomplished in two parts. The first evaluates the advantages of modelling the multicore architecture to estimate memory contention penalties and energy consumption. The second part of the validation evaluates the Deep Leaning functionality.

A. Multicore architecture validation

The first experiment of this validation isolates the memory contention penalty from the performance prediction, by executing various workloads in a homogeneous multicore cluster. All the nodes are set to a clock frequency of 3.4GHz. The experiment results of makespan and energy consumption for the different traces can be seen in Figure 7. The graphs compare the results of three different sets of data: *baseline* and *IRMaSim* represent simulation performed without and with the models presented in this article, *real* refers to the results of executing in a real cluster.

Its clear that the results given by the baseline model are very far from the real ones. Note that the makespan values produced have a relative error of about 37% in all traces. In contrast IRMaSim, as it models the performance penalty of memory contention, can reduce this error down to 3%. As can be seen in the figure, the energy results of the baseline model are also

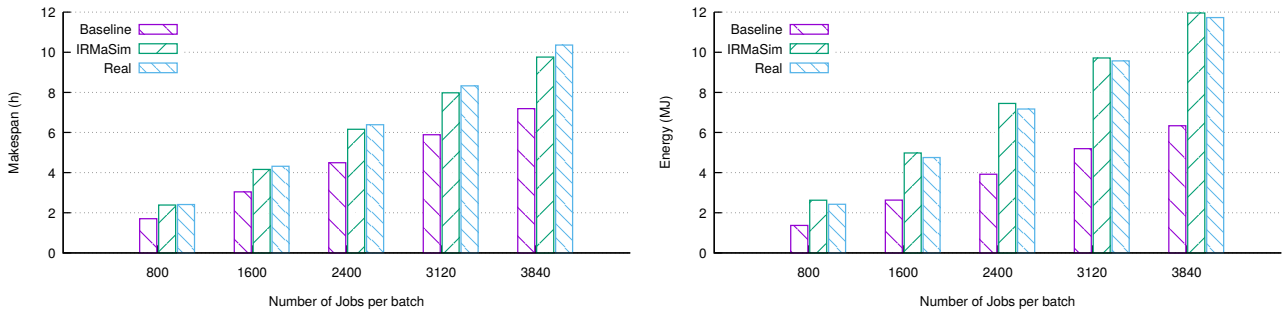

Fig. 8. Makespan and energy results in a heterogeneous cluster.

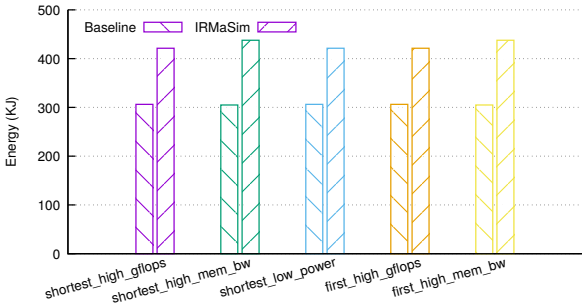

Fig. 9. Energy consumption of different scheduling policies.

disappointing, reaching a relative error of 47%. This is partly derived from the poor time prediction, but also because of the independent modeling of the energy consumption cores. Experiments show that the consumed power of the cluster reported by Batsim is 320W, which is significantly lower than that given by IRMaSim, 440W, or the real value, 400W. The improved power model of IRMaSim reduces the relative error to 10%.

A similar experiment has been done in a heterogeneous cluster, where out of the 15 nodes, 8 run at 3.4GHz, 4 at 2.5GHz and 3 at 1.7GHz. The makespan and energy results for this experiment are shown in Figure 8. As can be seen, these results are consistent with the previous experiment, which confirms that the heterogeneity modelling is also accurate. Indeed, the relative error of the baseline model is poor, with values of 30% and 45% in makespan and energy, respectively. And in contrast, IRMaSim substantially reduces both errors down to 5%. It is noteworthy that the errors decrease compared to the previous experiment, this is because the heterogeneous cluster includes low speed nodes that exhibit less performance penalty due to memory contention.

Another important advantage of IRMaSim is the fact that simulation time and energy consumption are very small compared to real execution. With speedups up to 200, simulating in a commodity computer, it allows researchers to make large number of simulations to test different approaches or scenarios.

B. Deep Reinforcement Learning validation

Once the validation of the simulator is accomplished, this section presents an illustration of the machine learning capabilities of IRMaSim. This serves two purposes, first it shows how a DRL agent converges to making correct decisions, and second proves the importance of correctly modeling the architecture of the cluster. Thus, a DRL agent is trained with a given trace on the heterogeneous cluster. This is done once with the baseline model, that does not regard the multicore architecture, and again with IRMaSim, that better models the cluster.

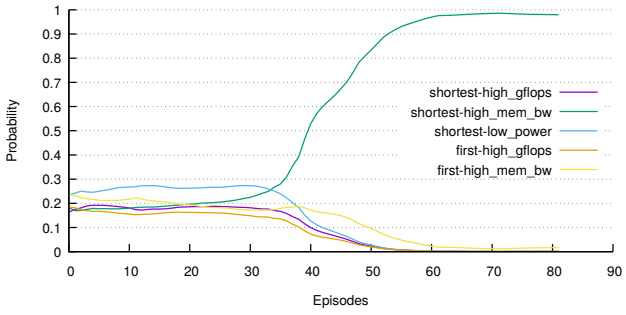

Fig. 10. Convergence of DRL agent toward chosen scheduling policy using the baseline model.

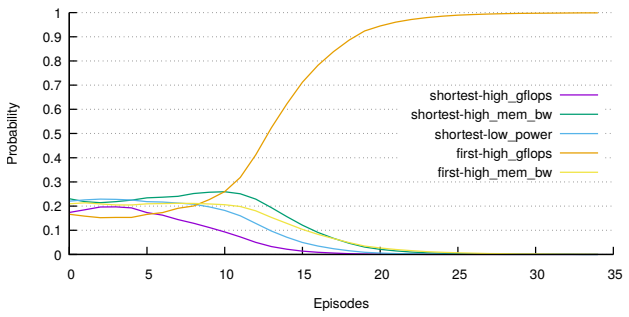

Fig. 11. Convergence of DRL agent toward chosen scheduling policy using IRMaSim model.

In this experiment, the agent is configured to minimise the energy consumption by choosing from a small subset of the policies presented in Table II. In order to determine if the agent makes the right decision, an execution was made with each policy, with the baseline and IRMaSim models. The energy consumption of the traces are shown in Figure 9.

When using the IRMaSim model, it appears that some policies are better than others. This is explained by the fact that the energy consumption of the multicore architecture benefits from the grouping of jobs in cores of the same processor. Then the policies that favor this grouping present an improved energy consumption. In contrast, the baseline model, where any active core consumes the same power, all policies present very similar results. Additionally, these results are in agreement with the previous experiments, where the baseline model gave substantially less energy consumption.

The learning progress of the DRL agent with successive training episodes in each scenario is shown in Figures 10 and 11. In both cases there is a point in the training, where the agent favors one policy over the rest. It can be seen that the chosen policy is in fact one that causes minimum energy consumption, according to the results in Figure 9. In the baseline case, in spite of the similarity of the results for each policy, the agent is capable of choosing a correct policy, even if it requires substantially more training episodes.

These results confirm that the machine learning techniques offer interesting possibilities to the workload management problem. But more importantly, they highlight the importance of the accuracy of the simulator. Because if the chosen policies were to be used in a real scenario, the one obtained with the baseline model would not be the best one, as it was chosen with a simplistic representation of the cluster.

VIII. Related Work

A first attempt to implement DRL techniques in task scheduling was to use the SLURM [31], which is a well known and established workload manager. However, several shortcomings were found. For example the plugin architecture of SLURM was too constrained, the launch script input options did not cover some necessary aspects like the memory bandwidth. In the end, for the kind of research aimed at, a simulation environment was preferred over physical computing infrastructure.

Other simulation frameworks were evaluated together with Batsim. Alea v4 [13] is developed on top of GridSim [7], implemented in Java and is open-source. It uses its own internal representation of machines (nodes), and provides a simple interface for creating management policies in Java.Scheduler Simulation Framework or ScSF [22] implemented as a wrapper around a real SLURM instance. Experiments are defined in a *controller*, which manages *worker* instances spawned in their own virtual machines. Components may be distributed, and physical network latencies have an impact in the simulation. New algorithms are implemented via SLURM plugins. Accasim [9] is an event-driven simulation. An *event manager* processes events within the simulated system, and a *dispatcher* assigns jobs to resources in the system. It is implemented in Python, with new algorithms integrated by extending base classes. These tools were disregarded in favour of Batsim due to a set of the following factors: scarcity of the documentation, difficulty of interaction with state-of-the-art DRL frameworks, overhead of managing a physical infrastructure and lack of data analysis tools.

In addition to classic job selection policies, like *First In First Out (FIFO)* and *Shortest Job First(SJF)*, there are several alternative approaches in the literature. Backfill schedules jobs based on their priority and the run times requested by the users, and then attempts to fill empty gaps in the schedule with lower priority jobs [17]. This approach that is sensitive to the accuracy of the requested run times has seen many improvements, like *EASY++* that tries to predict job run times [27]. In [11] the authors propose a machine learning approach, where they fit a L2-regularized polynomial model for predictions. Other authors also use *auto machine learning* frameworks to find the optimal model [24]. Other authors focus on optimising different metrics. For instance, ExpREsS focuses on minimizing the energy consumption as long as performance requirements are met [16].

On the other hand, another interesting approach is policy search. Unlike the previous techniques, the actual policy is inferred from both, the incoming jobs and the resource states. Genetic algorithms have been used successfully to optimize job sequencing [20], while reinforcement learning has also been used in [1] via enhanced *Q-learning* [28]. Most recent solutions leverage deep artificial neural networks for learning the optimal scheduling policy, such as DeepRM [15] and Wrangler [30]. RLScheduler [32] is a scheduler based on Reinforcement Learning that schedules batch jobs in a homogeneous cluster.

This body of work shows that the job scheduling and resource selection problems are decision problems that can benefit from machine learning techniques. Therefore there is a need for the development of new tools that can allow researchers to easily develop new ideas to address these problems. In this sense IRMaSim is a framework that aims to fill this void.

IX. Conclusion

This paper presents IRMaSim, a new open-source simulator that fills the void of solutions to develop ideas in the context of workload management, including those based on machine learning. Compared to other solutions, it extends the range of clusters that can be simulated by providing greater detail in the modeling of the architectures. On the one hand, it allows specifying different computational capabilities to the different nodes, allowing the definition of heterogeneous clusters. In addition, it simulates the contention derived from accessing shared resources, like memory. Also, it includes a model of power consumption more adequate for multicore architectures. Finally, it presents an easy to use API that allows the implementation of deep reinforcement learning techniques. This implies the design of agents, environments, observation and

action spaces, parameter tuning via data-analysis utilities and simulation log insight tools.

An experimental validation of the simulator is presented, that tests the new features against a real scenarios. Which include homogeneous and heterogeneous clusters, and traces extracted from different size execution batches. The results are substantially better than Batsim, as IRMaSim presents relative errors up to 5% in makespan and 10% in energy consumption.

The second part of the validation supports the possibility of employing Deep Reinforcement Learning techniques to the problem of workload management. It allows researchers to speed their research in the improvement of proposals and achievement of more sophisticated schedulers. But more importantly, it highlights the importance of leveraging simulation for this kind of research, and proves that the multicore architecture of clusters must be taken into account to obtain correct results

This is a young field and there are many exciting ideas that can be developed. Future lines of work can further refine the detail of the architectural modeling, as well as improve the framework by adding new policies, agents, observation and action spaces to enrich the experiments that can be performed.

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] M E. Aydin and E. Öztemel. Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33(2-3):169–178, 2000.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks—summary and preliminary results. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, page 158–165. Association for Computing Machinery, 1991.

[3] D. Bartuschat and U. Rüde. Parallel multiphysics simulations of charged particles in microfluidic flows. *J. Comput. Science*, 8:1–19, 2014.

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

[5] J.L. Bosque, P. Toharia, O.D. Robles, and L. Pastor. A load index and load balancing algorithm for heterogeneous clusters. *Journal of Supercomputing*, 65(3):1104–1113, 2013.

[6] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[7] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[8] P.F. Dutot, M. Mercier, M. Poquet, and O. Richard. Batsim: a realistic language-independent resources and jobs management systems simulator. In *Job Scheduling Strategies for Parallel Processing*, pages 178–197. Springer, 2015.

[9] C. Galleguillos, Z. Kiziltan, and A. Netti. Accasim: an hpc simulator for workload management. In *Latin American High Performance Computing Conference*, pages 169–184. Springer, 2017.

[10] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[11] E. Gaussier, D. Glesser, V. Reis, and D. Trystram. Improving backfilling by using machine learning to predict running times. In *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, page 64. ACM, 2015.

[12] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, and Y. Zhao. An efficient and fair multi-resource allocation mechanism for heterogeneous servers. *IEEE Transactions on Parallel and Distributed Systems*, 29(12):2686–2699, Dec 2018.

[13] D. Klusáček, S. Tóth, and G. Podolníková. Complex job scheduling simulations with alea 4. In *Proceedings of the 9th EAI International Conference on Simulation Tools and Techniques*, pages 124–129, 2016.

[14] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[15] H. Mao, M. Alizadeh, I. Menache, and S. Kandula. Resource management with deep reinforcement learning. In *15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 50–56. ACM, 2016.

[16] S. Maroulis, N. Zacheilas, and V. Kalogeraki. A holistic energy-efficient real-time scheduler for mixed stream and batch processing workloads. *IEEE Transactions on Parallel and Distributed Systems*, 30(12):2624–2635, Dec 2019.

[17] A.W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *IEEE trans. on parallel and distributed systems*, 12(6):529–543, 2001.

[18] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[19] B. Pérez, E. Stafford, J.L. Bosque, and R. Beivide. Energy efficiency of load balancing for data-parallel applications in heterogeneous systems. *Journal of Supercomputing*, 73(1), 2017.

[20] F Pezzella, G Morganti, and G Ciaschetti. A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

[21] O.D. Robles, J.L. Bosque, L. Pastor, and A. Rodríguez. Performance analysis of a CBIR system on shared-memory systems and heterogeneous clusters. In *Int. Workshop on Computer Architecture for Machine Perception, CAMP*, 2005.

[22] G.P. Rodrigo, E. Elmroth, P. Östberg, and L. Ramakrishnan. Scsf: a scheduling simulation framework. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 152–173. Springer, 2017.

[23] Thomas Röhl, Jan Eitzinger, Georg Hager, and Gerhard Wellein. LIKWID monitoring stack: A flexible framework enabling job specific performance monitoring for the masses. In *IEEE International Conference on Cluster Computing, CLUSTER2017, Honolulu, USA*, pages 781–784. IEEE Computer Society, 2017.

[24] M. Soysal, M. Berghoff, and A. Streit. Analysis of job metadata for enhanced wall time prediction. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–14. Springer, 2018.

[25] E. Stafford and J. L. Bosque. Improving utilization of heterogeneous clusters. *J. Supercomput.*, 76(11):8787–8800, 2020.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[27] D. Tsafrir, Y. Etsion, and D.G Feitelson. Backfilling using system-generated predictions rather than user runtime estimates. *IEEE Transactions on Parallel and Distributed Systems*, 18(6):789–803, 2007.

[28] C. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[29] Di Xu, Chenggang Wu, and Pen-Chung Yew. On mitigating memory bandwidth contention through bandwidth-aware scheduling. In *Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques*, PACT '10, page 237–248. Association for Computing Machinery, 2010.

[30] N. Yadwadkar. Machine learning for automatic resource management in the datacenter and the cloud. Technical Report UCB/EECS-2018-110, EECS Department, University of California, Berkeley, Aug 2018.

[31] Andy B. Yoo, Morris A. Jette, and Mark Grondona. Slurm: Simple linux utility for resource management. In Dror Feitelson, Larry Rudolph, and Uwe Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 44–60. Springer Berlin Heidelberg, 2003.

[32] Di Zhang, Dong Dai, Youbiao He, Forrest Sheng Bao, and Bing Xie. *RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning*. IEEE Press, 2020.