Encoding for Reinforcement Learning Driven Scheduling

Boyang Li[1], Yuping Fan[1], Michael E. Papka[2,3], and Zhiling Lan[1]

[1] Illinois Institute of Technology, Chicago IL 60616, USA
{bli70,yfan22}@hawk.iit.edu,lan@iit.edu
[2] Argonne National Laboratory, 9700 S Cass Ave, Lemont, IL 60439, USA
papka@anl.gov
[3] Northern Illinois University, 1425 W Lincoln Hwy, DeKalb, IL 60115, USA

Abstract. Reinforcement learning (RL) is exploited for cluster scheduling in the field of high-performance computing (HPC). One of the key challenges for RL driven scheduling is state representation for RL agent (i.e., capturing essential features of dynamic scheduling environment for decision making). Existing state encoding approaches either lack critical scheduling information or suffer from poor scalability. In this study, we present SEM (Scalable and Efficient encoding Model) for general RL driven scheduling in HPC. It captures system resource and waiting job state, both being critical information for scheduling. It encodes these pieces of information into a fixed-sized vector as an input to the agent. A typical agent is built on deep neural network, and its training/inference cost grows exponentially with the size of its input. Production HPC systems contain a large number of computer nodes. As such, a direct encoding of each of the system resources would lead to poor scalability of the RL agent. SEM uses two techniques to transform the system resource state into a small-sized vector, hence being capable of representing a large number of system resources in a vector of 100-200. Our trace-based simulations demonstrate that compared to the existing state encoding methods, SEM can achieve 9X training speedup and 6X inference speedup while maintaining comparable scheduling performance.

Keywords: Batch scheduling · Reinforcement learning· Scalability · High-performance computing.

1 Introduction

Cluster scheduler, also known as batch scheduler, plays a very critical role in high-performance computing (HPC). It is responsible for determining the order in which jobs are executed on a HPC system. Heuristic approaches are typically used for HPC cluster scheduling [1–3]. For instance, first come, first served (FCFS) is a well-known scheduling policy deployed on production HPC systems [1]. Backfilling is commonly used to enhance job scheduling by improving system utilization, where subsequent jobs are moved ahead to utilize free resources [1].

Recently, reinforcement learning (RL) is exploited to enhance cluster scheduling in HPC. In RL driven scheduling, the agent is trained to learn a proper scheduling policy according to a specific scheduling objective (e.g., reward) provided by system managers. Once trained, the agent can automatically interact with the scheduling environment and dynamically adjust its policy as workload changes. The application of reinforcement learning to the scheduling problem has yielded a number of promising results for cluster scheduling [4–12]. One of the key challenges for RL driven scheduling is *state representation (i.e., encoding)* for RL agent. Encoding must be *efficient* and *scalable*. An efficient encoding must capture critical system resource and waiting job information. These pieces of information must be represented as a fixed-sized input to the agent. RL agent is commonly built on deep neural network, and its computational cost grows exponentially with the size of its input. Production HPC systems typically contain a large number of computing resources. Hence, a direct encoding of each of the system resources leads to poor scalability of the RL agent. The training and inference time would increase drastically with the system size growing.

Existing encoding approaches can be broadly classified as *image-based* or *vector-based*. Image-based state representation uses a fixed-sized 2D image to represent waiting jobs and system resources: one dimension for resource availability and the other dimension for time duration [4–8]. Image-based state representation has two limitations. First, for HPC scheduling, although system resource has a fixed size, time duration could be infinite as a job may take seconds to days or even weeks to complete. As such, image-based state representation cannot effectively address the wide range of or even infinite time duration issue. Second, while image-based representation can capture resource requirement per job, it lacks the encoding of job information such as job priority or job waiting time [4–8]. In order to overcome the aforementioned issues of image-based encoding, vector-based encoding is widely adopted [9–12], in which the resource information and the job information are represented by vectors and concatenated into a fixed-sized vector input to the agent. Unfortunately, existing vector-based encoding methods suffer from two major shortcomings. First, prior vector-based encoding methods are mainly designed for directed acyclic graph (DAG) or malleable jobs, which are very different from rigid parallel jobs dominating in HPC with fixed resource requirements [9, 10]. Second, while several vector-based encoding methods are presented for HPC scheduling [11, 12], they either miss essential resource information [11], or lead to a huge state space, hence causing a scalability concern [12].

This work aims to tackle the aforementioned encoding problems. Specifically, we present SEM, a new **S**calable and **E**fficient state representation **M**odel for general RL driven scheduling in HPC. SEM adopts a vector-based encoding approach. It is designed to be *efficient* as it encodes system resources and waiting jobs, both being critical for efficient scheduling. It is also *scalable* such that it is capable of representing the scheduling state of a large-scale system in a small-sized vector (e.g., with 100-200 elements). RL agent typically uses deep neural network for decision making, and the computational cost of the neural work

grows exponentially with the input size. The significant reduction of input size via SEM encoding means dramatic scaling improvement of the agent.

In SEM, each waiting job is encoded into a 4-element vector to capture its state (i.e., job size, job estimated runtime, priority, and job waiting time). Rather than encoding each system resource individually as a vector, SEM encodes the system resources through the viewpoint of running jobs. We represent system resources by a concatenated vector of the resources used by a fixed number of running jobs. Resources used per running job is encoded as a 2-element vector (i.e. job size, estimated remaining time). This design is based on a *key observation*, that is, the resources belonging to the same job have the same system status (e.g., the same start and end time). Furthermore, parallel jobs occupying multiple computing resources are common in HPC. Hence, the number of running jobs is significantly less than the amount of system resources. In order to encode both system resources and waiting jobs into a small fixed-sized vector, we develop two solutions, one for systems with minimum job size requirement and the other for systems without such a requirement.

We compare SEM with existing encoding method under various configurations by using trace-based, event-driven scheduling simulation. Experimental results show that the use of SEM encoding can lead to a significant reduction in RL agent training and inference (or testing) time while maintaining comparable scheduling performance. Specifically, this paper makes three major contributions:

- We propose a novel encoding model for RL driven scheduler by leveraging the fact that the resources belonging to the same running job have the same system status and the number of running jobs is significantly less than the amount of system resources in parallel computing.
- We develop two methods to address the challenge that a fixed-sized state representation is required for RL agent even though the number of running jobs dynamically changes in a realistic environment.
- Extensive trace-driven experiments show that compared to existing state encoding models, our proposed method has a faster convergence speed. It allows the RL agent to achieve 9X training speedup and 6X inference speedup while maintaining comparable scheduling performance. Moreover, SEM can scale well as the system size increases.

The remainder of this paper is organized as follows. We start by introducing background and related work in Section 2. In Section 3, we describe SEM design. We present workload trace, comparison method, experiment setup and evaluation metrics in Section 4. The experimental results are presented in Section 5. Finally, we conclude the paper in Section 6.

2 Background and Related Work

2.1 Cluster Scheduling

A cluster scheduler is responsible for allocating resources and for determining the order in which jobs are executed on a HPC system. When submitting a

job, a user is required to provide two pieces of information: number of compute resources required for the job (i.e., job size) and job runtime estimate (i.e., walltime). The scheduler determines when and where to execute the job. The jobs are stored and sorted in the waiting queue based on a site's policy. Once a new job is submitted, job scheduler sorts all the jobs in the waiting queue based on a job prioritizing policy. A number of popular job prioritizing policies have been proposed, and a widely used policy is FCFS [1], which sorts jobs in the order of job arrivals.

In addition, backfilling is a commonly used approach to enhance job scheduling by improving system utilization, where subsequent jobs are moved ahead to utilize free resources. A widely used strategy is EASY backfilling which allows short jobs to skip ahead under the condition that they do not delay the job at the head of the queue [1].

2.2 Reinforcement Learning Driven Scheduling

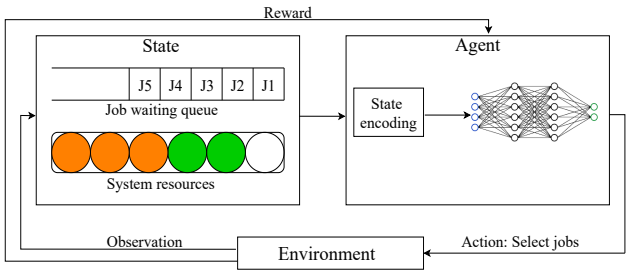


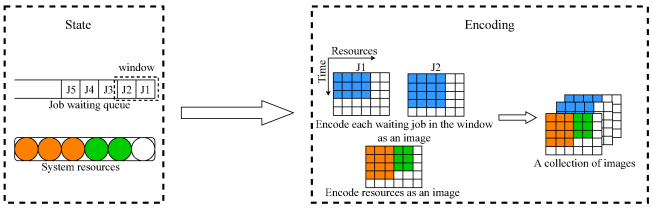
Fig. 1: Overview of RL driven scheduling. In the state, each system resource is represented by a circle. The circles sharing the same color indicate these system resources are allocated to the same running job; blank circles mean free/available system resources.

Reinforcement learning is an area of machine learning that is primarily focused on dynamic decision making where an intelligent agent takes actions in an environment with the goal of maximizing some reward [13]. The environment is captured by its state information. When the agent performs an action, it will be provided with the environment's response to that action as feedback (i.e., reward) and be taken to another state within the environment. These feedback and reward mechanisms allow the agent to learn, by trials and errors, how to act in order to maximize its reward [14].

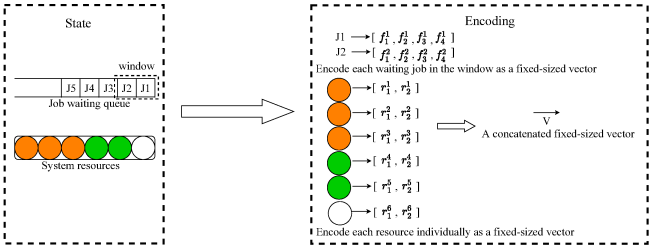
Pioneering studies have explored RL for cluster scheduling with encouraging results [4–12]. In RL driven scheduling, the agent is trained to learn a proper scheduling policy according to a specific scheduling objective (e.g., reward) provided by system managers. Once trained, the agent can automatically interact

with the scheduling environment and dynamically adjust its policy as workload changes. Since the state space is typically enormous, memorizing all states becomes infeasible. RL driven scheduling uses deep neural network for approximation [15]. Figure 1 shows an overview of a typical RL driven scheduling. At each step, state is observed and fed to the scheduling agent. The agent provides job selection and receives the reward as the feedback. It is obvious that state representation plays a critical role in RL driven scheduling.

2.3 Encoding Approaches



(a) Image-based state representation



(b) Vector-based state representation

Fig. 2: Overview of existing state representation approaches.

A key challenge of designing reinforcement learning driven scheduler is how to represent a dynamic scheduling state. Existing state representation models can be broadly divided into two classes: image-based state representation [4–8] and vector-based state representation [9–12].

Figure 2(a) illustrates image-based state representation. Image-based state representation typically represents the waiting job information and resource information by a fixed-sized 2D image: one dimension for resource availability and the other dimension for time duration [4–8]. The images for the first W waiting jobs are maintained for constraining the action space. Image-based representation has limitations of limited time duration and lack of the encoding of other critical job information as mentioned in Section 1 [4–8].

To overcome aforementioned issues, vector-based state representation methods are proposed [9–12]. Figure 2(b) illustrates vector-based state representation.

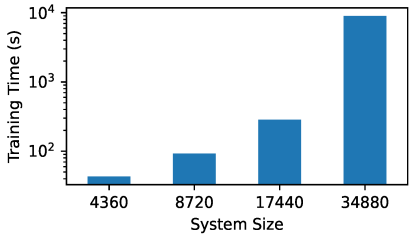


Fig. 3: Analysis of training costs with growing system size when using the vector-based encoding from the literature [12].

The waiting job information and resource information are represented by vectors and concatenated into a fixed-sized vector as the input to the neural network. Specifically, the first W jobs in the waiting queue are encoded for constraining the action space and each resource is also encoded as a vector. However, these proposed approaches still have their own limitations. In Decima and DL2, all the jobs were DAG jobs and could be decomposed into malleable tasks, whereas HPC was dominated by single rigid jobs that could not be decomposed [9, 10]. RLScheduler didn't capture the allocated resource state information in its state representation, hence missing essential resource information in the model building [11]. DRAS overcame these issues; however, the state encoding adopted in DRAS may lead to poor scalability [12]. DRAS used a vector to capture each system resource (e.g, compute node). As a production system may contain thousands of compute nodes, such an encoding results in an input vector in the size of thousands of elements and consequently leads to nontrivial training cost. As an example, we examine the potential training time when applying the vector-based state representation used in [12]. As shown in Figure 3, the cost exponentially increases with growing system size. The SEM model is designed to address these encoding issues for RL driven scheduling.

3 SEM Design

SEM, shown in Figure 4, is developed to provide a scalable and efficient state encoding for general RL driven scheduling in HPC. It encodes waiting jobs and system resources as vectors and concatenates them into a fixed-sized vector input to the agent. The design of SEM is based on the two key observation: (1) the resources belonging to the same job have the same system status and (2) the number of running jobs is significantly less than the amount of system resources. Rather than encoding each system resource individually as a vector, SEM encodes the system resources through the viewpoint of running jobs. We represent system resources by a concatenated vector of the resources used by a fixed number of running jobs. Such a resource encoding can significantly reduce vector size.

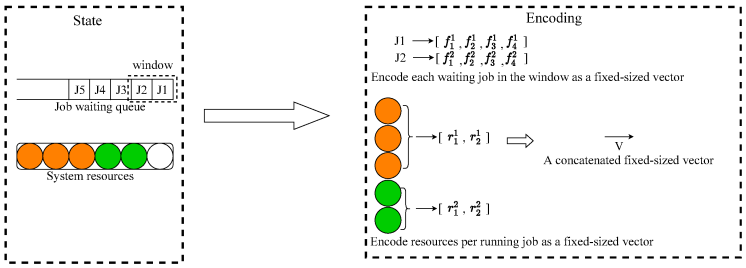


Fig. 4: SEM state representation. It captures system resources and user jobs as a fixed-size input to the RL agent depicted in the general RL driven scheduling shown in Figure 1.

We start by introducing the key observation through trace analysis in Section 3.1. In Section 3.2, we present the encoding of waiting jobs. We describe the resource encoding, including two methods to address the challenge that RL agent requires a fixed-sized input whereas the number of running jobs dynamically changes in a realistic environment, in Section 3.3. Finally, we analyze computational complexity of SEM in Section 3.4.

3.1 Observations

Table 1: Workload trace

Workload	Site	System Size	Period
Theta [16]	Argonne	4,360	Jan.2018-Dec.2019
Mira [17]	Argonne	49,152	Jan.2014-Dec.2014
Atlas [18]	LLNL	9,216	Nov.2006-Jun.2007
DataStar [18]	SDSC	1,664	Mar.2004-Apr.2005

For rigid jobs in HPC, the resources requested by the same job will be allocated to start the job and released after job completion as a whole (i.e the same start and end time). Thus, *the resources belonging to the same job have the same system status.*

To validate the assumption that the number of running jobs is much less than the system size, we analyze a number of workload traces from production supercomputers and four representative traces are presented in Table 1. Among them, Mira and Theta represent the HPC systems that have a minimum job size requirement, whereas Atlas and DataStar represent the systems without such a job requirement. Note that for DataStar and Atlas, their resources are expressed in the number of cores, hence the system size is set to the total number of cores for these systems [19].

We analyze these traces and present our data analysis in Figure 5. It plots the cumulative distribution function (CDF) of the number of running jobs. For example, while Theta has 4,360 compute nodes for default queue, the number of running jobs is no more than 32. Even for the larger system Mira with about 50K nodes, the number of running jobs is typically less than 50. In other words, we observe that *the number of running jobs is significantly less than the system size on production supercomputers*. Table 2 summarizes the maximum number of running jobs for each trace. There are two reasons for this phenomenon. First, a system is rarely fully occupied due to external and/or internal fragmentations [20, 21]. Second, parallel jobs using multiple system resources are common in HPC systems. Many capability computing systems such as those deployed for capability computing have a minimum job size requirement [22–24]. For instance, the minimum job size on Theta is 128.

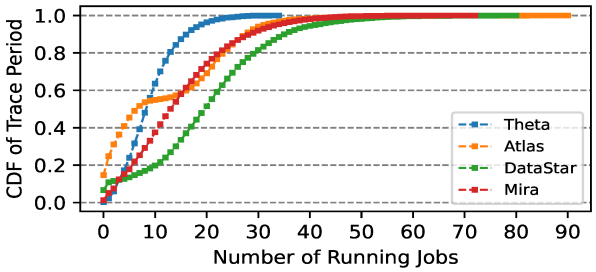


Fig. 5: The relationship between the number of running jobs and cumulative distribution function of trace period.

Table 2: Maximum number of running jobs

Workload	System Size	Maximum Number of Jobs
Theta	4,360	32
Mira	49,152	71
Atlas	9,216	90
DataStar	1,664	81

3.2 Encoding of Waiting Jobs

SEM adopts a vector-based state representation. It encodes the scheduling environment by capturing the state of both waiting jobs and system resources. In practice, the number of waiting jobs in job queue dynamically changes. When scheduling the queued jobs, since the available system resources are limited, usually only front jobs have chance to be selected for execution. As the RL agent requires a fixed-sized input, the first W jobs in the queue are encoded in previous vector-based methods [11, 12]. Take an example, we investigate the training

cost of DRAS agent under different W values [12], and the results are shown in Figure 6. In order to perform such an analysis, we purposely adjust the job density in the original trace to ensure there are sufficient waiting jobs in the queue. The results show that the training time increase is negligible (within 1s). One key reason is that compared to the system state, the waiting job representation accounts for a smaller percentage.

In this work, we use the same window based design. We choose to encode the first W jobs in the queue. For each waiting job, we encode it as a vector of four elements: job size, job estimated runtime, priority (1 means high priority; 0 means low priority), and job waiting time (time elapsed since submission). Job size is expressed by the percentage of system size.

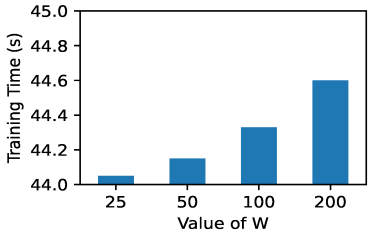


Fig. 6: Training cost of DRAS agent [12] on Theta trace [16] when using different values of W (a window of user jobs in the wait queue for decision making). One thousand jobs are used for training.

In general, W can be chosen by the system administrator. For typical HPC systems, W is set in the range of 10 - 100 [11, 12].

3.3 Encoding of System Resources

SEM encodes the system resources through the viewpoint of running jobs rather than encoding each system resource individually based on the key observations in Section 3.1.

Specifically, we separate the system resources into two groups: the resources occupied by running jobs, and the rest. For the former, SEM encodes them by the viewpoint of the running jobs. For instance, if a job is executed on c compute nodes, SEM encodes these c nodes by one vector: the group size (i.e., job size normalized by the system size) and the remaining occupancy time (i.e., the time difference between the job estimated end time and the current time). For the latter, SEM does not encode it because it is a redundant feature (which can be calculated by the system size minus the total allocated resources). Existing research indicates that redundant features add no useful information for model training [25]. By using such a resource encoding, SEM is capable of significantly reducing the input size. For example, for the 4,360-node Theta, instead of having

an input of over 4K features, SEM only needs to encode dozens of features — a reduction of two orders of magnitude.

A remaining issue is how many running jobs K should be encoded for representing the system state. RL agent requires a fixed-sized input whereas the number of running jobs dynamically changes. To attack this issue, we propose two methods. Note that production supercomputers can be broadly classified as systems with minimum job size requirement and systems without such a requirement. For the former systems, we propose a method named *zero-padding*. More specifically, we set K to the maximum number of running jobs which is calculated by dividing the system size over the minimum job size requirement. This number reflects the maximum number of concurrent running jobs on the system. For instance, Theta has 4,360 nodes for default queue and the minimum job size requirement in default queue is 128 nodes. Hence the maximum number of running jobs is 34 (4,360/128) on Theta. When the number of existing running jobs is less than K, we pad the rest of the vectors with zeros. Algorithm 1 shows the pseudo code of this zero-padding algorithm.

Algorithm 1 The zero-padding method

```
1: procedure System Resources Encoding(K)
2:     Read system state
3:     R ← number of running jobs
4:     Encode each running job as a vector
5:     if R < K then
6:         Pad the rest (K − R) vectors with zeros
7:     end if
8: end procedure
```

For the systems without minimum job size requirement such as Atlas and DataStar, the number of running jobs in practice is much less than the system size as shown in Table 2. We propose *K-largest-job* method. In the fields of natural language processing (NLP) and computer vision (CV), spatial pyramid pooling [26] and K max pooling [27] are shown to preserve the important information for variable size inputs. However, encoding for RL driven scheduling is different from that in the problems in NLP and CV. In NLP and CV, we cannot tell what information is important in advance. Fortunately, we can in RL driven scheduling. With respect to resource allocation, a large-sized job weights more than a small-sized job because the large-sized job requires a large amount of the system resources. Hence, when the number of running jobs is greater than K, SEM chooses to use the K largest jobs for encoding the system resources. Algorithm 2 shows the pseudo code of K-largest-job algorithm. K can be chosen by the administrator via experience or trace analysis.

To validate that the K-largest-job method can preserve system state with negligible information loss, we examine the percentage of total node-hour loss for each workload in Table 3. Node-hour is defined as the product of number of

Algorithm 2 The K-largest-job method

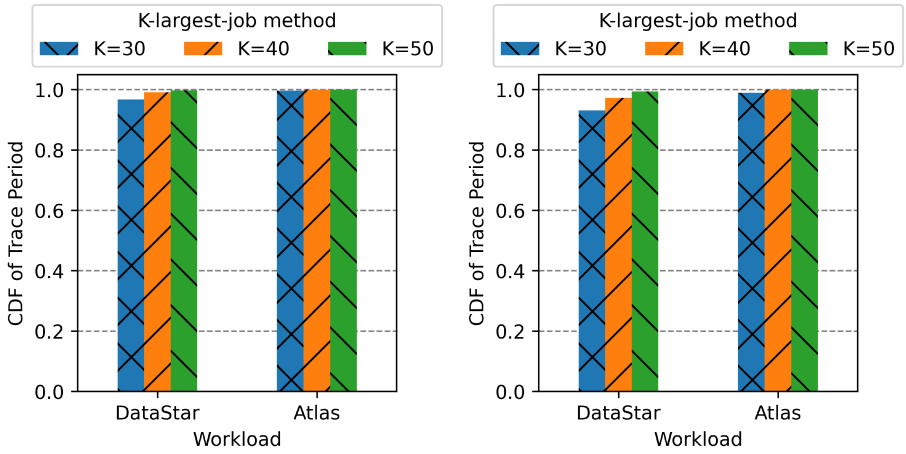
```
1: procedure System Resources Encoding(K)
2:     Read system state
3:     R ← number of running jobs
4:     if R > K then
5:         sort running jobs by job size
6:         Encode the first K largest jobs as vectors
7:     else
8:         Encode each running job as a vector
9:         Pad the rest (K − R) vectors with zeros
10:    end if
11: end procedure
```

nodes and time duration. For example, using one node for an hour is one node-hour. Node-hour is a commonly used resource allocation unit at supercomputing facilities [22,23]. The table shows the amount of node-hour losses by using different K values. It clearly indicates that total node-hour loss percentage is extreme trivial (within 1.3%), no matter when K is set to 30, 40 or 50 for both DataStar and Atlas workloads.

Table 3: Information loss when using the K-largest-job method

Workload	Information loss (node-hour loss %)		
	K=30	K=40	K=50
DataStar	1.22%	0.36%	0.01%
Atlas	0.23%	0.03%	9e-05

In addition, we also expect to ensure that the K-largest-job method can cover the majority of resources nearly all the time. Figure 7 shows the amounts of trace period that is covered by using different K values. Two plots are presented here, one for covering 90% of the allocated nodes and the other for 95%. When K is set to 30, for DataStar, out of thirteen months, it covers 90% of the allocated nodes for about 97% of the time and covers 95% of the allocated nodes for about 93% of the time. For Atlas, it covers 90% of the allocated nodes for about 99% of the time and covers 95% of the allocated nodes for more than 98% of the time. When K is set to 40, for DataStar, 90% of the allocated nodes are covered for about 99% of the time and 95% of the allocated nodes are covered for about 97% of the time. For Atlas, it can cover 95% of the allocated nodes for more than 99% of the time. When K is set to 50, for more than 99% of the time, 95% of the allocated nodes are covered for DataStar. For 100% of the time, 95% of the allocated nodes are covered for Atlas. This clearly demonstrates we can encode K largest running jobs to represent the system state with negligible information loss.



(a) 90% of the allocated nodes (b) 95% of the allocated nodes

Fig. 7: Trace coverage by using different K values

3.4 Complexity Analysis

For the existing vector-based encoding such as [12], the input vector is $O(W+N)$, where W is the number of waiting jobs and N is the size of the system. When adopting SEM, the input vector is reduced to $O(W+K)$, where K is the number of running jobs used for representing the system resources. As discussed in Section 3.1, K is greatly smaller than N. As shown in Figure 1, the number of neurons, hence the computational complexity of the agent, grow exponentially with the input size [28]. As a result, a significantly reduced-size state representation can lead to a great reduction of the corresponding RL model.

4 Evaluation

We describe how we evaluate our SEM design through extensive trace-based simulation using real workload traces collected from several supercomputers, and the results are listed in the next section.

4.1 Workload Trace

We implement SEM in the event driven scheduling simulator CQSim [29] and use it in our evaluation. We use two workload traces, Theta and DataStar, for evaluation. They are chosen to represent two HPC systems, where Theta stands for the computing system with minimum job size of 128 nodes and DataStar represents the computing system with no minimum job size requirement. All

experiments were conducted on a personal computer configured with Intel 2 GHz quad-core CPU with 16 GB memory.

For each trace, 60,000 jobs are used for training, 1,200 jobs for validation and 6,000 jobs for inference testing. During training, we monitor the convergence rate by taking a snapshot of the model after each episode (each episode consists of 600 jobs, thus there are total 100 episodes). The unseen validation dataset (1,200 jobs) is used for evaluating the trained model. Finally, an unseen set of 6,000 jobs is used for testing the RL agent and the results are shown in Section 5.

4.2 Comparison Methods

Several RL methods are presented for cluster scheduling such as deep Q-network (DQN), policy gradient (PG) [15, 30]. SEM changes the state encoding, and can be used with any RL method. In this study, we compare SEM with the vector-based encoding deployed in DRAS [12] when adopting the policy gradient method. Note that in the original DRAS state encoding, each system resource (i.e., compute node) is captured by a 2-element vector and the entire state encoding grows linearly with the system size. In the rest of the paper, we use SEM to denote the new state encoding presented in this work and DRAS to denote a general vector-based state encoding.

Table 4: RL agent configurations under SEM and DRAS

Configuration	Theta		DataStar	
	SEM	DRAS	SEM	DRAS
State Vector Size	268	8,920	280	3,528
Convolutional Layer	134	4,460	140	1,764
Fully Connected Layer 1	200	4,000	200	1,000
Fully Connected Layer 2	100	1,000	100	250
Output	50	50	50	50

4.3 Experiment Setup

For Theta, DRAS encodes the scheduling state in a 8,920-element vector where the 4,360-node machine is captured by 8,720 elements and the rest captures a window of W (=50) waiting jobs. When using SEM, K is set to 34 which is determined by the system size divided by the minimum job size requirement (i.e., 4,360/128). SEM uses the same sized window for waiting jobs. Following the reward deployed in DRAS [12], the reward is set to $w_1 \times \frac{\overline{t_i}}{t_{max}} + w_2 \times \frac{\overline{n_i}}{N} + w_3 \times \frac{N_{used}}{N}$, where $\overline{t_i}$ denotes the average wait time of selected jobs; t_{max} is the maximum wait time of jobs in the queue. Similarly, $\overline{n_i}$ is the average job size

of the selected jobs; N is the total number of nodes in the system; N_{used} is the number of occupied nodes. This reward function intends to balance three factors: to prevent job starvation, to promote capability (large) jobs, and to improve system utilization. The weights are equally set to 1/3.

For DataStar, DRAS encodes the scheduling state in a 3,528-element vector where the 1,664-node machine is captured by 3,328 elements and the rest captures a window of W (=50) waiting jobs. When using SEM, K is set to 40 according to the trace analysis in Figure 7. SEM uses the same sized window for waiting jobs. The reward is set to $\sum_{j \in J} -1/t_j$ [4], where J is the set of jobs currently in the system, t_j is the (ideal) duration of the job. This reward function aims to minimize the average job slowdown.

The details of SEM and DRAS neural networks for Theta and DataStar are listed in Table 4. For instance, on Theta, when using SEM, we use a convolutional layer with 134 neurons and two fully-connected layers with 200 and 100 neurons respectively. The output layer contains 50 neurons representing jobs in the window. When using the existing vector encoding [12], we use a convolution layer with 4,460 neurons, two fully-connected layers with 4,000 and 1,000 neurons respectively. The output layer contains 50 neurons representing waiting jobs in the window.

4.4 Evaluation Metrics

When evaluating different encoding methods, we use three quantitative metrics: *(1) agent convergence rate, (2) agent training/inference time, and (3) scheduling performance.*

Following the common practice, we use the following metrics to quantify scheduling performance. *Node utilization* measures the ratio of the used node-hours for useful job execution to the elapsed node-hours. *Job wait time* measures the interval between job submission to job start time. We analyze both average job wait time and maximum job wait time. *Job slowdown* denotes the ratio of the job response time (job runtime plus wait time) to its actual runtime. It is used to gauge the responsiveness of a system.

5 Results

In this section, we present the experimental results of comparing SEM with DRAS. Our analysis centers upon four questions:

1) Does SEM lead to a faster convergence rate for training RL agent? (Section 5.1)

2) Does SEM lead to comparable scheduling performance? (Section 5.2)

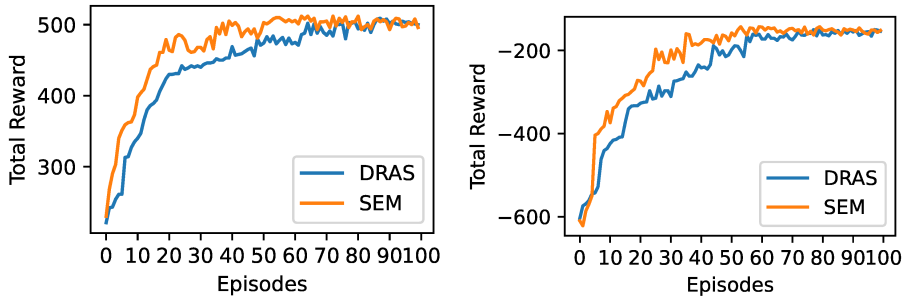
3) How much are RL agent training and inference costs when adopting SEM? (Section 5.3)

4) How scalable is SEM-enabled RL driven scheduling? (Section 5.4)

5.1 Convergence Rate

Convergence rate reflects how fast the scheduling agent can converge. We monitor the progress of the training by taking a snapshot of the model after each episode. We validate the trained SEM and DRAS enabled agent with an unseen validation dataset.

Figure 8 compares the convergence rates of the scheduling agents by using SEM and DRAS state representations separately. For Theta, SEM converges after 60 episodes while DRAS converges after 86 episodes. For DataStar, SEM converges after 53 episodes whereas DRAS converges after 74 episodes. It is clear that SEM leads to a faster convergence rate than DRAS. As shown in Table 4, SEM results in a much smaller sized neural network and consequently leads to faster convergence.



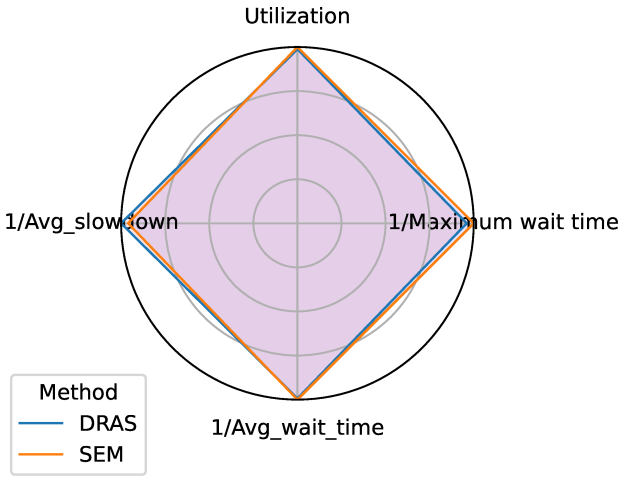
(a) Theta (b) DataStar

Fig. 8: Convergence rate of RL agent using different state encodings.

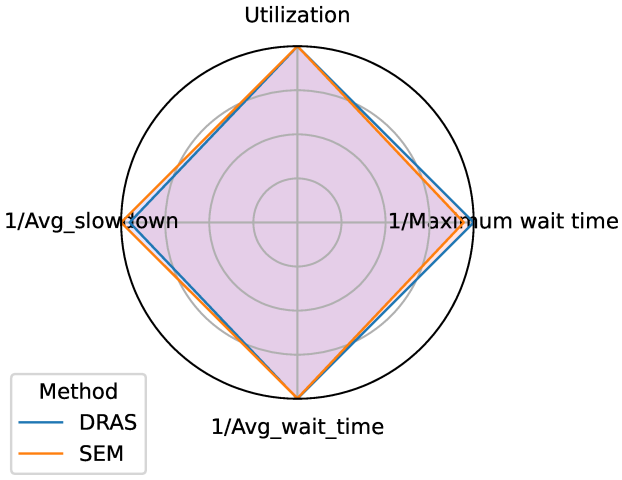
5.2 Scheduling Performance

Figure 9 presents the overall scheduling performance obtained by SEM and DRAS state representations with Kiviat graphs. We use node utilization, the reciprocal of average job wait time, the reciprocal of maximum job wait time and the reciprocal of average slowdown in the plots. All metrics are normalized within the range of 1. 1 means a method achieves the best performance among all methods. For all metrics, the larger the area is, the better the overall performance is.

We observe that the Kiviat graphs of SEM and DRAS almost overlap both on Theta and DataStar. On Theta, SEM performs slightly better than DRAS in utilization, maximum job wait time, average job wait time while DRAS performs a little better in average job slowdown. The difference of all these metrics are within 4%. On DataStar, SEM achieves a better performance than DRAS in utilization, average slowdown, average wait time whereas DRAS achieves a



(a) Theta

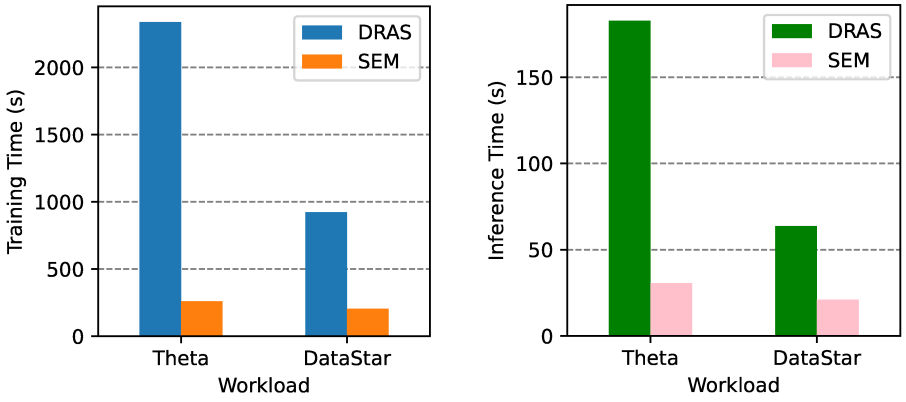


(b) DataStar

Fig. 9: Scheduling performance by using different state encodings. The plots use Kiviat graphs to provide a comprehensive view of scheduling performance. The larger the area is, the better the overall performance is. It indicates that SEM can achieve comparable scheduling performance as those obtained by the existing state encoding.

smaller maximum job wait time. The difference of all these metrics are within 6%. The results demonstrate that SEM can achieve comparable scheduling performance compared with DRAS and SEM can capture all the necessary system state information.

5.3 Training and Inference Speedup



(a) Training time (b) Inference time

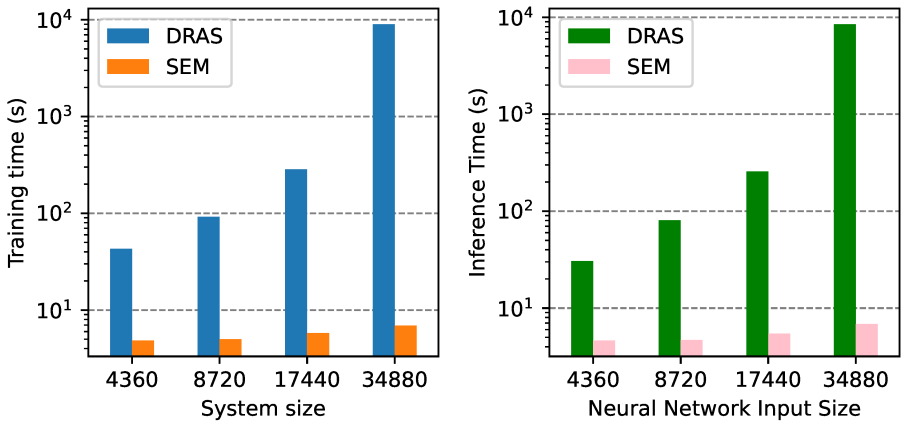
Fig. 10: Comparison of training and inference times.

In this subsection, we evaluate the training and inference time costs of RL agent by using different state encodings. The training time is the time cost of training 60,000 jobs and the inference time is the time cost of testing 6,000 jobs.

Figure 10 shows the training and inference time comparison between SEM and DRAS. The training and inference time of SEM is much less than that of DRAS. Compared to DataStar, SEM can achieve a larger training and inference speedup on Theta since Theta has a larger system size. The training and inference speedup can be up to 9X and 6X separately. This clearly shows SEM has a much faster training and inference speed than DRAS.

5.4 Scalability Analysis

As described in Section 2.3, existing vector-based encoding captures system resources individually. Since production supercomputers may contain a large number of resources to meet the ever-increasing demand of workloads, the use of existing vector-based encoding may lead to a neural network with millions or even billions of parameters. In contrast, SEM captures system state through a viewpoint of running jobs and is capable of capturing environment state using



(a) Training time (b) Inference time

Fig. 11: Analysis of training and inference times when system size increases. One thousand jobs are used for training and inference testing separately.

a fixed-sized vector of dozens of elements. Such a significant reduction of input size via SEM encoding means dramatic scaling improvement of the agent.

In this set of experiments, we examine training and inference time when increasing system size by using SEM and DRAS state encodings. The results are presented in Figure 11. In this figure, system size is increased from 4,360 to 34,880. As system size increases, number of jobs and vector input also grow proportionally in this experiment. The plots show that the training and inference time of SEM increases very slowly while the training and inference time of DRAS increases drastically with the system size increasing. When the system size is set to 34,880, the use of existing vector-based encoding leads to a training cost of over 8,000s, whereas SEM results in a training cost of only 7s. This means over 1000X improvement compared to DRAS state encoding.

6 Conclusions

Reinforcement learning driven scheduling is a promising approach for cluster scheduling. Prior RL driven scheduling studies mainly focus on exploiting different RL methods with little attention to state representation. Existing state representation approaches either lack sufficient scheduling information or suffer from poor scalability. In this work, we have presented a new and generic state representation called SEM for general RL driven scheduling. SEM captures the state of scheduling environment in a fixed-sized vector. It is efficient as it captures both waiting jobs and system resources for scheduling environment. It is also scalable as it utilizes a new method to capture system state through a viewpoint of running jobs. In this study, two methods are presented

to encode scheduling environment into a fixed-sized vector, i.e., zero-padding and K-largest-job. Experimental results show that SEM can lead to a faster convergence speed, up to 9X training/inference time reduction, and high scalability as compared to the existing encoding methods.

The proposed SEM encoding is generally applicable to a variety of RL driven scheduling methods that use deep neural network for decision making. In this study, we focus on single type of resource encoding. Part of our future work is to expand SEM for multi-resource scheduling.

7 Acknowledgement

This work is supported in part by US National Science Foundation grants CCF-2109316, CNS-1717763, and CCF-2119294. This research used data generated from resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357.

References

1. A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.
2. W. Allcock, P. Rich, Y. Fan, and Z. Lan, "Experience and practice of batch scheduling on leadership supercomputers at argonne," in *Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP)*. IEEE, 2017.
3. B. Shahzad and M. T. Afzal, "Optimized solution to shortest job first by eliminating the starvation," in *The 6th Jordanian Inr. Electrical and Electronics Eng. Conference (JIEEEC 2006), Jordan*, 2006.
4. H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016.
5. G. Domeniconi, E. K. Lee, V. Venkataswamy, and S. Dola, "Cush: Cognitive scheduler for heterogeneous high performance computing system," in *DRL4KDD 19: Workshop on Deep Reinforcement Learning for Knowledge Discover*, vol. 7, 2019.
6. B. Baheri and Q. Guan, "Mars: Multi-scalable actor-critic reinforcement learning scheduler," *arXiv preprint arXiv:2005.01584*, 2020.
7. R. L. de Freitas Cunha and L. Chaimowicz, "Towards a common environment for learning scheduling algorithms," in *2020 28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2020, pp. 1–8.
8. B. Ryu, A. An, Z. Rashidi, J. Liu, and Y. Hu, "Towards topology aware pre-emptive job scheduling with deep reinforcement learning," in *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, 2020, pp. 83–92.
9. H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.

10. Y. Peng, Y. Bao, Y. Chen, C. Wu, C. Meng, and W. Lin, "Dl2: A deep learning-driven scheduler for deep learning clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 1947–1960, 2021.
11. D. Zhang, D. Dai, Y. He, F. S. Bao, and B. Xie, "RLScheduler: an automated HPC batch job scheduler using reinforcement learning," in *SC'20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE/ACM, 2020.
12. Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, and M. E. Papka, "Deep reinforcement agent for scheduling in HPC," in *Proceedings of the 35th International Parallel and Distributed Processing Symposium*. IEEE, 2021.
13. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
14. L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.
15. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
16. Theta. [Online]. Available: https://www.alcf.anl.gov/theta
17. Mira. [Online]. Available: https://www.alcf.anl.gov/alcf-resources/mira
18. PWA. [Online]. Available: https://www.cs.huji.ac.il/labs/parallel/workload/
19. D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.
20. W. Tang, Z. Lan, N. Desai, D. Buettner, and Y. Yu, "Reducing fragmentation on torus-connected supercomputers," in *2011 IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2011, pp. 828–839.
21. F. A. P. Pinto, L. G. L. de Moura, G. C. Barroso, and M. M. F. Aguilar, "Algorithms scheduling with migration strategies for reducing fragmentation in distributed systems," *IEEE Latin America Transactions*, vol. 13, no. 3, pp. 762–768, 2015.
22. Argonne Leadership Computing Facility (ALCF). [Online]. Available: https://www.alcf.anl.gov
23. Oak Ridge Leadership Computing Facility (OLCF). [Online]. Available: https://www.olcf.ornl.gov/
24. Lawrence Livermore National Laboratory. [Online]. Available: https://www.llnl.gov/
25. A. Appice, M. Ceci, S. Rawles, and P. Flach, "Redundant feature elimination for multi-class problems," in *Proceedings of the 21st International Conference on Machine Learning*, 2004, p. 5.
26. K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.
27. N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," *arXiv preprint arXiv:1404.2188*, 2014.
28. J. Heaton, *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.
29. Cqsim. [Online]. Available: https://github.com/SPEAR-IIT/CQSim
30. D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.