Deep Reinforcement Learning based Elasticity-compatible Heterogeneous Resource Management for Time-critical Computing

Zixia Liu
Department of Computer Science,
University of Central Florida
Orlando, Florida
zixia@knights.ucf.edu

Liqiang Wang
Department of Computer Science,
University of Central Florida
Orlando, Florida
lwang@cs.ucf.edu

Gang Quan
Electrical and Computer Engineering Department,
Florida International University
Miami, Florida
gang.quan@fiu.edu

ABSTRACT

Rapidly generated data and the amount magnitude of data analytical jobs pose great pressure to the underlying computing facilities. A distributed multi-cluster computing environment such as a hybrid cloud consequently raises its necessity due to its advantages in adapting geographically distributed and potentially cloud-based computing resources. Different clusters forming such an environment could be heterogeneous and may be resource-elastic as well. From analytical perspective, in accordance with increasing needs on streaming applications and timely analytical demands, many data analytical jobs nowadays are time-critical in terms of their temporal urgency. And the overall workload of the computing environment can be hybrid to contain both time-critical and general applications. These all call for an efficient resource management approach capable to apprehend both computing environment and application features.

However, the added up complexity and high dynamics of the system greatly hinder the performance of traditional rule-based approaches. In this work, we propose to utilize deep reinforcement learning for developing elasticity-compatible resource management for a heterogeneous distributed computing environment, aiming for less occurrences of missing temporal deadline while maintaining low average execution time ratio. Along with reinforcement learning we design a deep model employing Long Short-Term Memory (LSTM) structure and partial model sharing for multi-target learning mechanism. The experimental results show that the proposed approach could greatly outperform baselines and serve as a robust resource management for variant workloads.

CCS CONCEPTS

• **Software and its engineering → Cloud computing**; • **Theory of computation → Scheduling algorithms**; **Reinforcement learning**.

ICPP '20, Aug. 17-20, 2020, Edmonton, Canada

ACM ISBN 978-1-4503-8816-0/20/08…$15.00
https://doi.org/10.1145/3404397.3404475

KEYWORDS

deep reinforcement learning, time-critical, heterogeneous, elasticity, multi-cluster, distributed computing

ACM Reference Format:
Zixia Liu, Liqiang Wang, and Gang Quan. 2020. Deep Reinforcement Learning based Elasticity-compatible Heterogeneous Resource Management for Time-critical Computing. In *49th International Conference on Parallel Processing - ICPP (ICPP '20), August 17–20, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3404397.3404475

1 INTRODUCTION

Nowadays' distributed computing resources hosting data analytical jobs are often organized in the unit of cluster, where each cluster represents a closed association of computing nodes that a data analytical work can be carried out in a distributed manner [21]. To further expand the computing resources in a large scale, opting for a multi-cluster computing resource raises its benefit and necessity due to the following reasons: Firstly, the multi-cluster environment can be naturally derived from the geographical distribution of resource. Secondly, it sometimes can be beneficial to organize overall resource into clusters as a separation for managing jobs and data. Thirdly, the facility may utilize online computing resources from service providers as additional clusters for resource expansion. Examples of a multi-cluster environment could happen when an institution has several physical clusters at its branches (may at different locations). Another example could be a hybrid-cloud, which may consist of multiple private (self-owned) and public cloud clusters. In accordance, an efficient resource management being aware and compatible with such a multi-cluster computing infrastructure should be presented to guide job distribution.

Moreover, there are other computing environment features to be taken into considerations. Firstly, it is possible that computing nodes in different clusters have different computing capabilities. In other words, multiple clusters composing the environment could be heterogeneous. Secondly, certain clusters could possess elasticity. Here, elasticity is referring to that the capacity of the cluster could be temporally expanded as desired to fit the expanded computing demands, which is more often observed in cloud computing as one of its main features [15]. It is certainly beneficial if a proposed resource manager could be compatible with such elasticity capability and be aware of heterogeneity.

Furthermore, job features are equivalently important. For example, besides general analytical jobs, many jobs nowadays could

be streaming jobs, which aim to timely process gradually arriving streaming data, and can be regarded as conducting repetitive executions with different batch of data. Such jobs are intrinsically end-to-end delay-sensitive, thus are time-critical. In a multi-cluster environment, considering differentiated transmission overhead for a time-critical job, it requires differentiated temporal execution deadline on different clusters, which should be regarded as a job feature and be considered by the resource management approach.

All computing environment features such as multi-cluster, elasticity and heterogeneity, and the job features such as job type and timeliness, add up to the complexity of generating a satisfactory resource management approach that can well utilize the multi-cluster environment. Thus, it is very hard to devise a comprehensive rule-based approach. Moreover, the high dynamic feature of the system status as well as the desire for a timely online scheduling decision hinder the utilization of iterative searching based approaches. In vision of this, we propose to utilize deep reinforcement learning (DRL) techniques in this work to obtain a resource management that could fulfill the expectation. The major contributions of this paper include:

- We propose a deep reinforcement learning based approach utilizing **LSTM model** and **multi-target regression with partial model sharing mechanism**, and compare its effectiveness with respect to other baseline and RL approaches.
- The DRL-based resource management is designed for distributed multi-cluster computing environments with considering its **heterogeneity** and being **elasticity-compatible**.
- The DRL-based resource management provides scheduling support for **time-critical** (delay-sensitive) computing in such a multi-cluster environment as described.

The following sections of this paper are organized as follows: In Section 2, we introduce the background and related works. Then, the problem description and the reinforcement learning based approach are presented in Section 3 and Section 4 respectively. In Section 5, we discuss the experiment results. And finally, the conclusion is given in Section 6.

2 BACKGROUND AND RELATED WORK

Reinforcement learning (RL) has recently gained astonishing accomplishments in a diversified range of tasks such as artificial intelligence, object tracking, vehicle management, robot navigation and dialogue system [16][22][17][11][6][7]. Its advantages are well demonstrated in extracting knowledge from continual interactions with the environment even in complicated systems, which is otherwise hard to be abstracted by rule-based approaches even under expert guidance. Once trained, it could respond fast to inference for providing timely action decisions in contrast to searching-based approaches. Such capabilities of both learning from complexity and fast response, make it a good choice for solving intricate problems such as the one we are tackling with in this work.

In accordance with this consensus, reinforcement learning has been adopted to deal with decision-making problems in a distributed computing environment. [14] presents "DeepRM" that utilizes reinforcement learning for obtaining a resource manager to coordinate workload execution in a cluster for improving execution efficiency. [23] uses multi-agent reinforcement learning in obtaining a method aiming at the job scheduling problem in a Grid computing environment, in purpose of realizing load balancing. In [3], a cloud controller towards resource allocation for applications in a cloud environment as an automatic workflow is obtained via reinforcement learning where techniques in accelerating training process are integrated. [1] applies Q-learning in training towards optimal policy for dynamic resource allocation for applications in a cloud. [10] tries to solve a joint virtual machine resource allocation and power management problem by proposing a hierarchical framework learned via reinforcement learning. In [9], a model-free approach is obtained by deep reinforcement learning, targeting at minimizing average end-to-end tuple processing time for distributed stream data processing systems. [2] presents DRL-Cloud, a resource provisioning and task scheduling system achieved via deep reinforcement learning focusing on reducing energy cost.

[12] proposes a resource management approach for time-critical applications in a distributed computing environment via reinforcement learning. It is the most related one to this work. However, it differs with this work in multiple aspects. Firstly, the underlying problem is largely different. Although similarly dealing with a multi-cluster environment, [12] does not consider cluster heterogeneity and elasticity, both of which are considered in this work. Later descriptions in Section 3 and 4 will reveal how greatly this will change the problem nature and increase problem complexity. Secondly, depending on the new problem feature, the most important action value in this paper's reinforcement learning is vitally redefined with novel consideration and insights. Thirdly, the added problem complexity leads to a brand new model structure design. [12] uses a general neural network with fully-connected layers, while we propose a deep neural network based on LSTM structure and a partial network sharing multi-target learning mechanism. Fourthly, our experiments provide more thorough observations and show comparison with approach in [12]. Hence, this work shows great importance and significant difference comparing to [12] from aforementioned aspects.

In short, although sharing similar concept of utilizing reinforcement learning in decision-making tasks related to distributed computing, these related works are all different from our work in this paper due to different problem characteristics and goals.

3 PROBLEM DESCRIPTION

The intended global resource management is aiming to **(1) reduce occurrences of missing temporal deadline events** while **(2) maintaining a low average execution time ratio** for a hybrid workload containing multiple time-critical and general jobs, by properly scheduling them to appropriate computing clusters in the underlying computing environment. We depict the overall architecture of the problem in Figure 1 and present detailed problem description as follows.

The underlying computing environment is composed of multiple computing clusters. For each cluster, its overall computing resource is expressed as the number of executors it could provide. The "executor" here is a basic resource allocation unit containing a combination of virtual CPU and memory resources, which is adopted in popular resource managing frameworks such as YARN [21], and utilized by computing environments such as Apache Hadoop and Spark. Different from [12], where executors are assuming to be identical

among different clusters, cluster heterogeneity is allowed in this work. That is, different clusters in the computing environment may have diversified executors with different computing capabilities. A heterogeneity factor (the larger the stronger) will be assigned to each kind of such executors, representing its relative computing capability. Furthermore, in this paper, clusters are allowed to have elasticity, that is, their computing capability in terms of executor numbers, can be temporarily expanded (with an upper bound) when necessary, to fit workload pressure. This assumption coordinates with the trending of elastic computing resources.

The holistic workload contains a number of continually arriving jobs, where each job can be classified into one of the three categories[12]: **streaming jobs** (Cate-1), **non-streaming time-critical jobs** (Cate-2) and **other general non-time-critical jobs** (Cate-3). As aforementioned, streaming jobs can be regarded as conducting repetitive executions with data batches and are inherently delay-sensitive therefore time-critical. Such a categorization of jobs helps manage hybrid workload in presenting a large range of representative user analytical needs.

Arriving jobs first enter the global job buffer. For each moment t when global job buffer is non-empty, given one job in the global buffer, the intended global resource manager selects a proper cluster in the computing environment for execution. Here, how to derive an effective approach to fulfill the global resource management is the main problem.

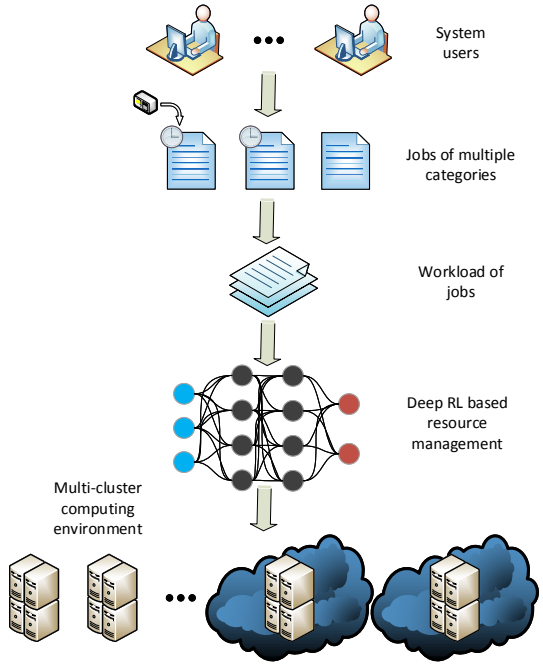


Figure 1: An illustration of the problem architecture.

This problem is inevitably complex. Firstly, abundant information, such as job and computing environment features and system dynamic need to be considered. Secondly, a job can have cluster-specific difference, such as cluster-specific deadlines due to differentiated transmission overhead and cluster-specific execution time due to cluster heterogeneity. Cluster elasticity could also change system resource status and thus affect job execution. The model should be able to handle these issues and provide good performance to both management goals. Solely attending to partial of the available information or partial of the objectives can lead to unsatisfactory performances. Integrated balancing between goals and short-term sacrificing yet long-term benefiting scheduling actions are potentially desired. The extreme difficulty lays in whether, when or how such kind of trade-off should be made, which is quite uncertain for rule-based models and brings favor to DRL-based approaches.

4 THE DEEP REINFORCEMENT LEARNING BASED APPROACH

Our goal is to obtain an efficient resource management approach with a neural network model via deep reinforcement learning. In this section, we state our elaborative considerations in accommodating the problem to model design and training skills, starting by a brief introduction to reinforcement learning technique.

4.1 Introduction to Reinforcement Learning

Reinforcement learning is a mechanism that enables model improvement through continual interaction with the application environment defined upon several key concepts: environment, state, episode, action and reward (or value).

Environment describes the overall world where the actual state transitions happen. **State** is used to express the environment status at a moment. Based on the adopted representation, the state s_t at moment t can potentially contain historical or instantaneous information of the environment. **Action** represents the set of actions performable to environment at the moment. The instant **reward** is a quantitative incentive feedback that the model receives from environment at a moment, whereas the **value** is a form of the accumulated reward (optionally decayed) observed in a longer duration. Definition of the value of taking an action a in state s under a decision-making policy π can be formulated as below [18]:

$$V^{\pi}(s,a) = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s, a_t = a\right\}$$

where E_{π} represents the expectation under policy π, r_{t+k+1} is the instant reward at moment $t+k+1$, a_t is the action taken at moment t and γ is the decay factor. An **episode** is a round of 'game' that marks the reaching of the termination state.

Reinforcement learning enables model improvement via the general interaction mechanism as depicted in Figure 2: Based on environment state s_t at t, an action is decided by the model and an environment state transition consequently happens. Over the time, model improvement can be carried out by learning from collected interactive experience in purpose of maximizing action values. Such a process is repeated multiple times until the accomplishment of training process.

4.2 Reinforcement Learning Method Design

In this work, **environment** is the entire computational system where the overall processing of jobs take place. The **action** set contains number of actions equal to the number of clusters forming

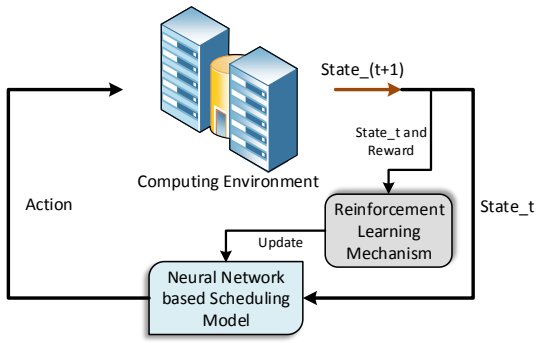


Figure 2: An illustration of reinforcement learning.

the computing infrastructure, where each action corresponds to deploying the job to that specific cluster. An **episode** is the whole process of scheduling and finishing the execution of a workload consisting of a certain number of jobs.

Environment **state** is composed of two main components: computing system state and scheduling job information. Computing system state (iterative for each cluster) includes static cluster features like its sequence number, elasticity information (including cluster's normal capacity and maximum capacity under expansion), heterogeneity factor (hetero factor) and dynamic features like cluster occupation status in a past time interval and current accumulated total occurrences of missing deadline events in cluster.

For scheduling job information, jobs in different categories can possess different attribute sets. We utilize streaming job (Cate-1) information as a super set for description, and other job categories can adjust to this with unambiguous accommodations. Such information includes job category, expected hetero factor, standard execution time with respect to using executors with the expected hetero factor, execution deadline, total duration, discrete resource request distribution and its heterogeneity sensitivity measure to describe job's adaptation capability to clusters with different hetero factors. We explain some of the keywords here as below:

Heterogeneity sensitivity measure: A job's execution time is often proportionally affected by cluster's hetero factor if the factor is within a given range. In this work, a job's heterogeneity sensitivity measure '*sen*' (integers in [1, 5] in experiment) is used to represent such range by $R = [h - sen \cdot \zeta, h + sen \cdot \zeta]$, where h is the job's expected hetero factor and $\zeta = 10$ in experiment. If the cluster's hetero factor belongs to R, this job's execution time in cluster will be proportionally affected from its standard execution time by the cluster hetero factor. Otherwise, if the cluster's hetero factor is outside the range, this job will cease to further fully reflect change in its execution time. This corresponds to job behaviors in practice where its execution time could be affected accordingly by executors within a range of hetero factors, yet will not fully receive benefit (or harm) in execution time for too large hetero factor changes, due to other execution overheads.

Job's cluster-specific difference: Job can have cluster-specific difference. Such as, job's execution deadline for each cluster can be different, possibly due to cluster-specific data transmission overheads from data source, when regulating a uniform end-to-end batch execution time. It could also be due to job's program transmission overhead to cluster before starting the execution. Job execution time can also be cluster-specific due to cluster heterogeneity. To lower user profiling burden, instead of requiring cluster-specific execution times, our model only requires the standard execution time with respect to (w.r.t.) job's expected hetero factor and can handle heterogeneity even without the complete execution time information.

Discrete resource request distribution: A streaming job may have resource request fluctuation during its repetitive executions. Analogous to [12], when executing in a cluster, it is affiliated with an length 10 array to describe its discrete resource request distribution, with each position accounting for 10 percentage possibility. For example, if such a distribution array containing eight 30s and two 60s, it means in each execution of the job, it has 0.8 probability to request 30 executors and 0.2 probability to request 60 executors in this cluster.

Table 1: State representation in our deep reinforcement learning model for 5 clusters

Vector Component	**Dimensions**
Cluster ($i = 1 \ldots 5$)	
Cluster sequence number	1
Normal capacity	1
Maximum capacity if elastic	1
Cluster heterogeneity factor	1
Occupation status (latest 105 steps)*	$105 \longrightarrow 150$
Current total missing deadlines	1
	775 (155×5)
Job	
Category	1
Expected heterogeneity factor	1
Heterogeneity sensitivity	1
Discrete resource request distribution	10×5
Standard execution time	1×5
Execution deadline	1×5
Duration	1
	64
Overall state vector	**839** (775 + 64)
Only * row includes temporal information	

The overall composition of the state representation is presented in Table 1. For cluster occupation status, we would like to use the records roughly in the latest 100 steps. In experiment, the latest 105 steps is used per input design desire, and this information is specially traversed to create a 150 dimensional vector for each cluster. Details of the traverse will be stated in Section 4.4.

The influence of deploying a job onto a cluster is destined to be long-term due to its execution duration. Meanwhile, since performance metrics related to number of missing deadline events and execution time statistics have significant response delay from occurring moments of their most influential contributing factors such as inappropriate deployment or resource competition within cluster, it is difficult to define the instantaneous action reward at any moment. In this vision, we alternatively define the value of an action, concentrating on action's long-term influence. In this

work, the **value** $v^{(j)}$ of an action that schedules a job j to a cluster at some moment, is defined as follows:

$$v^{(j)} = \frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{-\eta_j} \left[M_j + \sum_{t=t_s}^{t_e} \beta^{D_t} \left(W_j^{(t)} + W_{cl}^{(t)} \right) \right] - \psi_{ih} \cdot \psi_{ic} \cdot R_j$$

Here, η_c and η_j are the heterogeneity factor of the cluster and the expected heterogeneity factor of the job, respectively. M_j is the number of missing deadline events of job j where resource waiting does not happen at execution beginning. Note that a job can have more than one missing deadline events, such as, since a streaming job can be repetitively executed for multiple times. $W_j^{(t)}$ records the happening of each missing deadline event of job j at moment t, if it does not belong to M_j. Similarly, $W_{cl}^{(t)}$ records the number of missing deadline events of all jobs in the cluster at moment t if resource waiting happens at their execution beginning. t_s and t_e correspond to the deployment and termination moment of job j, respectively. β is the decay factor, and D_t records how many new jobs have been deployed to the cluster after t_s, till moment t. R_j represents the overall average execution delay ratio of job j (its average execution time divided by standard execution time, then minus 1 for representing delay). In the experiment, weighted factors are applied to the components of $v^{(j)}$, we omit its showing in the formula for simplicity.

m_{ih} and m_{ic} follow Eq. 1 of m_Ω, where Ω services as a place holder for 'ih' or 'ic'. The events corresponding to 'ih' and 'ic' are Improper_Heterogeneity (IH) and Initial_Competition (IC), respectively. Here, IH refers to the case where an improper cluster arrangement is caused by the scheduling action, that the cluster heterogeneity factor is too low for the job, causing execution deadline violation. IC refers to where resource competition (resource waiting) happens right after the job's deployment to the cluster designated by the scheduling action. Similarly, ψ_{ih} and ψ_{ic} follow Eq. 1 of ψ_Ω, respectively. Here, $P_{m_\Omega} \geq 1$ and $P_{\psi_\Omega} \geq 1$.

$$m_\Omega = \begin{cases} P_{m_\Omega} & \text{isEvent} \\ 1 & \text{Otherwise} \end{cases} \quad \psi_\Omega = \begin{cases} P_{\psi_\Omega} & \text{isEvent \& } R_j > 0 \\ 1/P_{\psi_\Omega} & \text{isEvent \& } R_j \leq 0 \\ 1 & \text{Otherwise} \end{cases} \tag{1}$$

The design of the value formula is originated from the consideration that when resource competition contributes to the causing of the missing deadline event of job j, the root cause of the resource competition can be complex. It could be due to the deployment of job j on a specific cluster or later deployment of other jobs on the same cluster. To alleviate such intertwined influence when deriving a clearer action evaluation, all missing deadline events of job j related to resource competition will be decayed over numbers of newly arriving jobs after j. Furthermore, the number of overall missing deadline events at any moment from all jobs in the cluster which could be caused by resource competition is also added following the same decay pattern, in purpose of attending the potential mutual influence of j from and to other coexisting jobs in the cluster. On the contrary, other missing deadline events of job j without resource waiting will not be decayed over time. The factor $\frac{\eta_c}{\eta_j}$ in formula helps the model discern the potential differentiated effects of cluster heterogeneity to the job. Such heterogeneity influence is also considered in R_j due to its definition. The factors $m_{ih}, m_{ic}, \psi_{ih}$ and ψ_{ic} work as a whole to assist the proper avoidance of certain irregular behaviors which the model should avoid.

4.3 DRL Model Structure and Decomposition of Value Definition

According to Table 1, the model input (RL system state) contains two types of information: **Non-temporal information** describing static or accumulated properties of job and computing environment, and **temporal information** related to computing environment occupation status in an interval of past moments. There exist neural network structures more specialized in dealing with temporal sequential information, such as Long Short-Term Memory (LSTM). LSTM has shown effectiveness in various tasks, such as speech recognition, music modelling and language translation [5] [24]. We plan to utilize the LSTM structure to process the temporal information portion of the input.

Also, the action value is a vital feedback signal for RL process. In fact, the neural network in our approach is directly performed as a value estimator. How well the action value can be estimated will undoubtedly affect performance in a large extent. Nonetheless, the value definition contains multiple components for better depicting action influence and such constitution adds up the estimation complexity. It makes the changing behavior of the integrated action value more difficult to be predicted. And, the possibly differentiated numerical range of components and their variation patterns could vanish influence of some individual component, which may affect learning and the multi-goal optimization result.

A plausible solution is to consider components in the value definition as multi-objectives in reinforcement learning, converting it to a Multi-Objective Reinforcement Learning (MORL) problem [20]. In a MORL problem, the value space can be composed of multiple dimensions, *i.e.,* $\mathbf{V}^\pi(s,a) = \left(v_1^\pi(s,a), \ldots, v_m^\pi(s,a)\right)$. With such formulations, our approach can be described as a scalarized[19] single-policy[4, 13] learning algorithm for MORL. More specifically, a linear scalarization function is utilized to regulate a united measure over the vector of the value components, and provide a single scalar value feedback for a single policy learning problem. The scalarization mechanism in this work can be described as [20]: $\hat{SV}_{linear}(s,a) = \sum_{i=1}^{m} w_i \times V_i(s,a)$, where $V_i(s,a)$ is intended to be a value estimator for v_i^π and w_i are weights. Now the problem becomes how to achieve multiple value estimators $V_i(s,a)$, one for each value component of our problem.

Based on the refined problem nature, the purpose of estimating multiple real value component outputs via training with the same training data can be modeled as a multi-target learning (regression) problem. Accordingly, we decide to utilize a partial network sharing skill to potentially facilitate the process. It uses multiple networks sharing the front a few layers, where one network aims for one of the expected value component estimators. Here, multi-target learning focuses on simultaneously training networks as value components estimators desired by the formulated scalarized single-policy MORL algorithm with potential intertwine, where the shared layers serve as a joint structure aiming to capture abstract input encoding shareable and beneficial to all networks. This model construction and learning mechanism enable the value estimation at its individual component level.

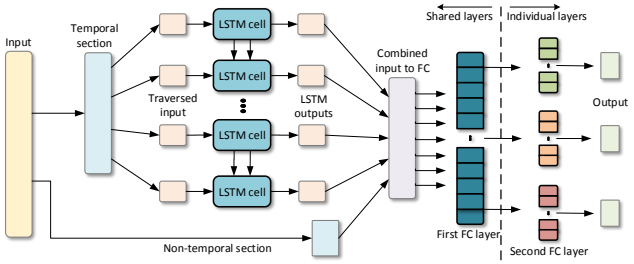


Figure 3: The structure of our deep neural network.

The overall network structure of this work is depicted in Figure 3. As illustrated, the input is separated into temporal related and non-temporal related sections. The temporal section will first go through the LSTM-based structure and the aggregated temporal hidden states will be integrated with the non-temporal section in the original input to enter the fully-connected (FC) layers. The LSTM session network and the first FC layer are the shared layers, whereas the second FC layer is isolated for each estimator, three in total, corresponding to each of the following value components:

$$v_1 = -\frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{\eta_j}\left(M_j + \sum_{t=t_s}^{t_e} \beta^{D_t} W_j^{(t)}\right)$$

$$v_2 = -\frac{\eta_c \cdot m_{ih} \cdot m_{ic}}{\eta_j}\left(\sum_{t=t_s}^{t_e} \beta^{D_t} W_{cl}^{(t)}\right)$$

$$v_3 = -\psi_{ih} \cdot \psi_{ic} \cdot R_j$$

Here v_1 focuses on feedback related to missing deadline events of job j itself, v_2 focuses on mutual influence of missing deadline events of all jobs on-board the cluster, and v_3 focuses on average execution delay ratio of job j. The integrated value of an action is then decided based on the (weighted) sum of outputs of the three network estimators.

4.4 Training Enhancement Skills

We apply several training enhancement skills that are suitable for the model training of the underlying problem with the potential to increase training efficiency and performance. They are:

Cluster occupation status traverse:

The cluster occupation status in the last 105 time steps is specially traversed to generate the actual input of the LSTM session of the model, which also forms the temporal portion of the model input. This can be seen as a segmentation process of the original status vector, aiming at transforming it into a series of status segments (possibly with overlaps) following the same temporal order, each as a continual sub-sequence of the original one. Such kind of segmentation helps preserve the original temporal series information to be captured by LSTM structure, while revealing occupation evolution by the sub-sequence in each segment available to the LSTM cell.

For each cluster, as depicted in Figure 4, the process starts at the beginning of the occupation status vector with a 10-element increment for each segment's start-point and with length 15. Therefore, each two consecutive segments have a 5-element overlap. Overall, the segmentation for the length 105 status vector results in 10 segments of length 15, concatenated as a 150 length vector. For totally 5 clusters as in the experiment, we have a 750 length vector as the actual temporal portion of input.

Training with decayed learning rate: We utilize Adam optimization [8] for learning rate tuning. Here we denote it as $Adam(\alpha)$, where α is a base learning rate. Our exemplary experiments reveal that a relatively large base learning rate, such as $Adam(1e-2)$ or $Adam(1e-3)$ may adversely affect training performance. And a single base learning rate with a smaller α shows relatively promising influence, such as $Adam(1e-4)$. However, for these approaches, no manual decay in the base learning rate is included.

We decide to further supplement a manually decayed base learning rate to Adam. More specifically, for the first 1000 training episodes in the experiment, the learning rate is $Adam(1e-4)$, and afterwards, it becomes $Adam(1e-5)$. Such Adam optimization with a manually decayed base learning rate provides the potential to combine the benefit of more swift change in the early stage and more fine tuning in the later stage of the training.

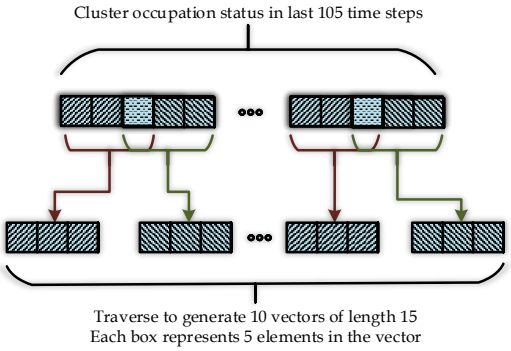


Figure 4: Traverse of cluster occupation status.

We've also adopted some training skills in [12], shown to be effective for training RL-based resource management, including:

Training with randomized workload: Randomness is added to important job feature variables to inject variations of the workload in each episode. This stimulates better state space exploration and pressurizes the network in continually refining itself towards generalized knowledge of its duty to suit workload variations.

Modified ϵ-greedy exploration: A rule-based baseline model is supplemented to guide the action perturbation for exploration. When a random action perturbation is needed (by probability ϵ from the original ϵ-greedy exploration), it then has another probability that this perturbation will be instead provided directly by the supplemented baseline, otherwise, a normal random action perturbation is performed. This is in purpose of letting a rule-based model inject its better-than-random knowledge to partially guide the general exploration, such that exploration efficiency is increased.

Solving multi-job selection dilemma: The proposed resource management can schedule one job at a moment. So, besides deciding the scheduling action for a job, it also needs to supplementarily select a targeted job for its scheduling, if more than one exist in the global buffer. To accomplish this goal, designing a model to accept all jobs in the global buffer at once is difficult due to the absence

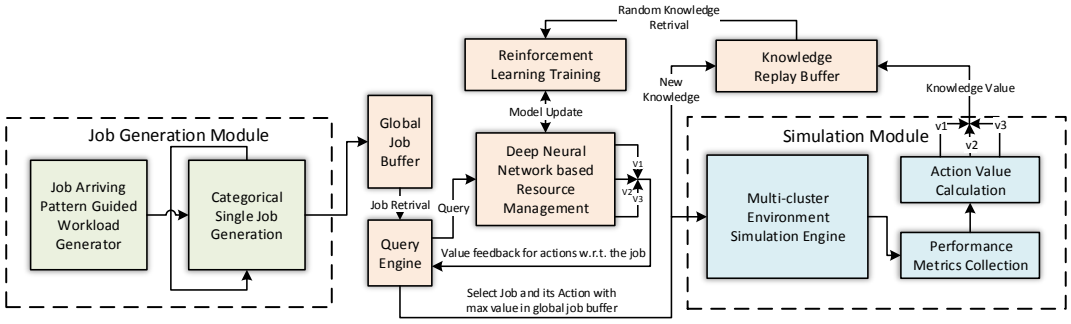


Figure 5: Training architecture of our deep neural network in one episode.

of job count upper bound and the further exacerbated exploration difficulty. Therefore in the adopted design, our model still accepts a single job input at once, but an outer level iterator will traverse the global job buffer where an overall value vector would be assembled to provide job and its action selection in one integration.

The overall training architecture is shown in Figure 5.

5 EXPERIMENTS

The experiments are conducted via simulation. In the experiments, the testing distributed computing environment is set as composed of 5 clusters, with executor capabilities for each described by the vector: [800, 1200, 1500, 1800, 2300]. Notice that here executors among different clusters can be heterogeneous. Accordingly, the heterogeneity factor for each cluster is: [70, 60, 100, 80, 90]. For elasticity, we suppose two of the clusters: the 3rd cluster and the 5th cluster have elasticity and their resource variation behaviors are managed by the elasticity controller as described below.

Elasticity controller: Executor capacity of an elastic cluster can be expanded by an increment amount (100), when the cluster occupation ratio in any moment of a past time window (100 steps) is $\geq$ 95%. The accumulative expansion in effect can be no larger than a maximum limit (200). On the other hand, if no occupation $\geq$ 95% is detected in the past time window, the expansion capacity in effect will be returned in stages by a decrement amount (100).

Note that our approach focuses on cluster-level global resource management. To accomplish job execution, a local intra-cluster scheduler is still needed. Our approach, instead of replacing or intervening, will actually cooperate with the local scheduler and as a benefit has the potential to coordinate with different local schedulers. We use a popular generic local scheduler for experiment.

Local intra-cluster scheduler: At each moment in a local cluster, if there are jobs halting and awaiting for resource from previous moments, they are prioritized for resource satisfaction. After that, job requests to start execution at this moment are considered for resource allocation. Whenever queuing is needed, the same sequence as the job arrival is followed. If a job's resource need cannot be satisfied, it will be put into halting (resource waiting), and try again in the next moment. Here, resource satisfaction means that the job's executor request can be fully fulfilled by the cluster.

To compare with our deep RL-based approach, multiple rule-based baselines are considered:

Random (RAN): Jobs are deployed to one of the clusters forming the multi-cluster computing environment solely by randomness.

Round-Robin (RR): Jobs are deployed to each cluster forming the multi-cluster computing environment in a round-robin manner, starting from a random one at the very beginning.

Most Available First (MAF): Cluster with averagely most available computing capacity in a past time window relatively for the considered job will be selected for job deployment. Here the available computing capability for cluster i is defined as: $(1 - RO_i) \times CAP_i/ER_i$, where RO_i, CAP_i and ER_i are the average occupation ratio of cluster i in the past time window, current total capacity of cluster i (elastic capacity in effect is included) and the most likely executor amount request from the job to cluster i, respectively.

For all baselines, the job with the least amount of worker request in the global job buffer will be selected. Similar as before, the worker request of the job is defined as the average of the number of executors most likely to be requested by the job with respect to each cluster. In this way, the MAF baseline roughly mimics the idea of 'SF-E', which is the best baseline as presented in [12].

Additionally, we also demonstrate our DRL approach by comparing to **another RL model (denoted as RL-FC)**, which roughly follows the reinforcement learning approach in [12].

From the computing environment perspective, the comprehensive job submission (arriving) pattern to the infrastructure could affect system status changing and thus be relevant to system state transition. In experiments, we use three different stochastic job arriving patterns that are utilized in [12] with the name Uniform, Bernoulli and Beta for thorough approach testing. However, different from [12], where models are trained separately for each job arriving pattern, in this work, the intended model is solely trained with the Uniform job arriving pattern, and its generality of direct usage on other job arriving patterns will be presented. The possibility of having i as the time interval between job arriving events for Uniform pattern is: $P(i) = \frac{1}{|b-a+1|} I_{i \in [a,b]}(i)$ with parameter setting $a = 1$ and $b = 33$. I is the indicator function. We refer readers to [12] for other supplemental information about job arriving patterns and according formulas for Bernoulli and Beta patterns.

Definition of performance metrics:

Recall there are two major goals for our resource management: (1) reducing number of missing deadline events during job executions, and (2) maintaining low job execution time ratio. These are equivalent as simultaneously minimizing two metrics:

TMDL: Total number of occurrences of missing deadlines for all jobs in all clusters during the execution of the workload.

AJER: Average job execution time ratio among all clusters, *i.e.,*

$$AJER = \sum_{j=1}^{N_w} \left(\frac{100 \cdot \sum_{k=1}^{n_j} \left(\frac{AR_{jk}}{SR} \right)}{n_j \cdot N_w} \right)$$

where SR and AR_{jk} are the standard execution time of job j and the execution time of job j in its k-th running when executing in the designated cluster, respectively. n_j is the number of executions for job j and is usually larger than 1 for Cate-1 jobs. N_w is the number of total jobs in each episode. Note that by dividing AR_{jk} (relevant to cluster heterogeneity) with SR (irrelevant to cluster heterogeneity), the resulting AJER is influenced by heterogeneity of the selected cluster, thereby including the cluster heterogeneity influence into evaluation. For cluster elasticity, its influence is also included in both TMDL and AJER implicitly.

Further, an integrated score $\boldsymbol{S_{log}}$ is defined by caring for the comprehensive performance in both goals.

$$S_{log} = sign(S) * \log_{10}(\max(|S|, 1)) \text{ as } S = -\text{TMDL} + 50 * (100 - \text{AJER})$$

For $S_{log} \in \mathbb{R}$, the higher the better. And a score 0 for S_{log} can be roughly seen as equivalent to having an average execution time same as the standard one and with no missing deadline events. The weighting factor 50 in S is for balancing between two goals, since TMDL is an accumulative measure for all jobs in the workload, whereas AJER is an averaged measure.

To facilitate qualitative comparison, three more comparative measures are defined: Fully-dominant (F), Semi-dominant (S) and Non-dominant (N), which mean that in a testing episode, whether the RL approach has better performance comparing to MAF, for both, only one, or none of the performance metrics (TMDL and AJER in consideration), respectively. For 50 testing episodes we used for approach comparison, the **F/S/N** scores are given as a distribution among all testing episodes. Therefore, for performance preference, F>S>N. Before discussing the experiment results, some training parameters are provided in Table 2.

Table 2: Some training parameters

Notation	Description	Value
E	Number of training episodes	1500
N_w	Number of jobs in each episode	1000
K	Capacity of knowledge buffer	50000
B	Training batch size	2000
L_I	Model input layer size	839
L_C	Input size to a LSTM cell	15
N_{fc1}	Neurons in first FC layer	800
N_{fc2}	Neurons in second FC layer (identical size for three networks)	200
L_O	Output layer size (identical size for three networks)	5

Approach Performance:

Consequently, the performance comparison (in S_{log}) for our approach (notated as RL-LSFC) with respect to other baseline approaches RAN, RR, and MAF, in different training episodes during the training process, is shown in Figure 6. For each vertical comparison, all models are included to compare for 50 testing episodes. The same workload is used for all models in one testing episode, but is re-generated for each testing episode.

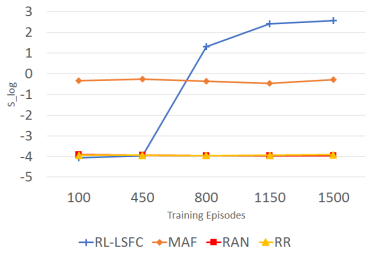
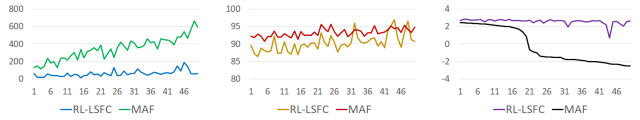


Figure 6: Performance comparison (S_{log}) of our RL approach RL-LSFC and baseline approaches in different training episodes.

From it, we can observe that among baselines, Random (RAN) and Round-robin (RR) provide similar performance and are significantly surpassed by MAF. For our RL-LSFC, it gradually improves itself during training and surpasses all baselines during the mid of overall episodes, and continues to increase its performance towards the end of training. A detailed performance comparison between the final RL model after training and MAF is shown in Figure 7 and the average statistics are shown in Table 3.



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)

Figure 7: Comparison of RL-LSFC and MAF for 50 testing episodes. Three sub-figures are w.r.t. TMDL, AJER and S_{log}. Data in all figures are sorted uniformly in descendent by S_{log} of MAF for viewing convenience. (L) Lower is better. (H) Higher is better.

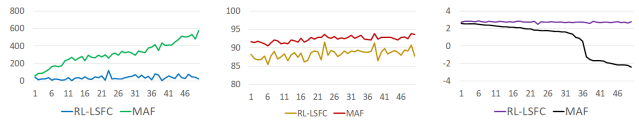
Table 3: Statistics w.r.t. Figure 7

	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**61.70**	**90.30**	**2.57**	46/4/0
MAF	343.84	93.22	−0.28	-

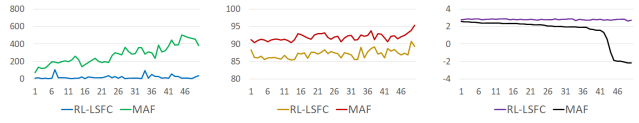
We can see that our approach trained via deep reinforcement learning outperforms MAF in a large scale. For TMDL, it reduces the occurrence of missing deadline events by 5.57 times. For AJER, it shortens average job execution ratio by 2.92%, thus obtains a significant higher S_{log} score with 2.57 comparing to −0.28. As well, RL-LSFC achieves excellent 46/50 Fully-dominant, 4/50 Semi-dominant and no Non-dominant in 50 testing episodes.

In fact, during experiment, a phenomenon is observed that the model obtained by our deep RL approach can perform equivalently well or even better for workloads that is statistically "less stressful" than the ones in training. Here, a coarse definition of the "stress" can be described as how crowded the computing environment is during the peak period of the workload execution. The workload "stress" can be changed via the Uniform pattern parameter b. In training, we intentionally select $b = 33$, which could reasonably stress the computing environment nearing to its maximum, yet not over-saturate the overall computing capability. And the obtained RL-LSFC could consequently be utilized in a wide-variety of workload scenarios that is "less stressful" than training to the computing environment. Therefore, although RL-LSFC has already shown

promising performance in $b = 33$ scenario, we are more caring for its performances in other "less stressful" scenarios, as they represent a much broader variety of applicable cases. We test such generality by two testing scenarios, one with a stress-reduced workload ($b = 36$), and the other with a more stress-reduced workload ($b = 40$). The results are shown in Figure 8 and Table 4.



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)



(d) TMDL (L) (e) AJER (L) (f) S_{log} (H)

Figure 8: Comparison of RL-LSFC and MAF in variant workloads. (a)-(c) are related to $b = 36$ scenario. (d)-(f) are related to $b = 40$ scenario. Other instructions are the same as Figure 7.

Table 4: Statistics w.r.t. Figure 8

(a)-(c)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**37.66**	**88.32**	**2.73**	50/0/0
MAF	311.44	92.35	0.87	-
(d)-(f)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**19.76**	**87.12**	**2.79**	50/0/0
MAF	276.1	91.95	1.55	-

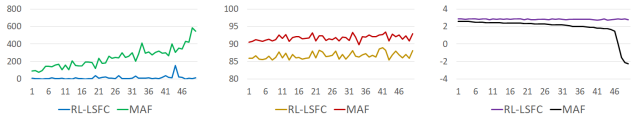
We can see that as the workload gradually becomes less stressful, RL-LSFC performs even better, and is consistently fully-dominant with respect to the MAF baseline in all testing episodes for both scenarios. Since the stress of the computing environment could largely vary in daily usage, the generality of the RL model to fit for such variation is certainly an apparent advantage.

Performance in other job arriving patterns

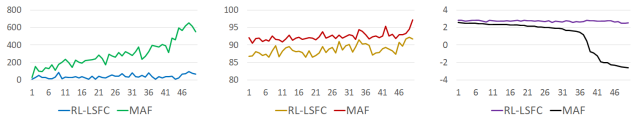
Different from [12], where a RL model is trained for each job arriving pattern, we solely train one deep RL model based on the Uniform pattern, and present that the obtained model could work equivalently well for other job arriving patterns directly, which is an out-of-intuition but exciting result. To accomplish, we directly utilize the obtained RL-LSFC model with Uniform pattern onto other two patterns, Bernoulli and Beta. The results in Figure 9 and Table 5 show that RL-LSFC with respect to the Uniform pattern indeed can work well directly with the other two job arriving patterns. We envision that this may because the obtained model is capable to capture intrinsic information from the provided system status for deciding scheduling actions, and is less prone and less sensitive to job arriving pattern shift. The observed irrelevancy of job arriving patterns in experiments is a great supplement towards approach generality.

Comparison with a fully-connected layers model RL-FC

In this experiment, we would like to compare models of our approach to a RL model with fully-connected layers based structure, which roughly follows the ideas in [12] (denoted here as RL-FC).



(a) TMDL (L) (b) AJER (L) (c) S_{log} (H)



(d) TMDL (L) (e) AJER (L) (f) S_{log} (H)

Figure 9: Comparison of obtained RL-LSFC and MAF in other job arriving patterns. (a)-(c): Bernoulli pattern. (d)-(f): Beta pattern. Other instructions are the same as Figure 7.

Table 5: Statistics w.r.t. Figure 9

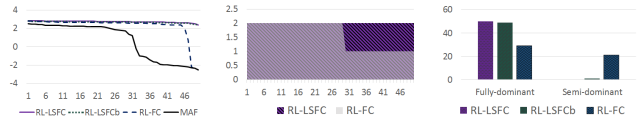
(a)-(c)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**13.32**	**86.67**	**2.81**	50/0/0
MAF	243.18	91.75	1.92	-
(d)-(f)	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC	**39.14**	**88.68**	**2.71**	50/0/0
MAF	295.66	92.41	1.11	-

To strengthen the comparison, besides RL-LSFC, we intend to additionally supplement another deep RL model variant also obtained by our approach. Actually, by the greatly separated recognition of individual objectives of our MORL problem empowered by the approach architecture, we are able to achieve variants of RL models with different balancing between objectives such as via weight adjustment of objectives forming the scalarized value feedback signal. With such benefit, we obtain another variant of our model, namely RL-LSFCb. It shares general conception with RL-LSFC, but with a slightly different balancing between objectives.

Three RL approaches, RL-LSFC, RL-LSFCb and RL-FC together with the MAF baseline are put into 50 testing episodes, results of which are shown in Figure 10 and Table 6. We can see that although the RL-FC model achieves slightly better TMDL than RL-LSFC, it pays a too large cost in AJER and has thus lower score in S_{log} than both of our models. Similar weakness of it can also be observed in F/S/N distribution comparing to both of our models. Thus, the RL-FC model in experiment shows weaker balancing ability between multiple optimization goals and a weaker overall performance. Meanwhile, two of our models, RL-LSFC and RL-LSFCb show differently prioritized yet better performance. RL-LSFCb could surpass RL-FC in all aspects, which indicates potential advantage of our approach. But its balancing in dual-objectives causes a hit in comprehensive score S_{log} comparing to RL-LSFC and promotes the RL-LSFC model as the best performing model here. The best is, our approach grants the freedom to choose either of them per user preference.

It is worth mentioning that we are not intending to declare an absolute structural or approach advantage here, since a direct and thorough competition of two RL approaches requires extensive hyper-parameter searching and tuning, which is not the main objective and beyond the scope of this paper. Nonetheless, the exemplary RL comparative experiments here show that our approach presents

decent capability in balancing multi-objectives while providing good overall scheduling performances.



(a) S_{log} (H) (b) Dominant area (c) F/S/N distribution

Figure 10: Comparison of three RL models w.r.t. MAF. (a) each curve is independently sorted for viewing convenience. In (b), we give F:2, S:1 and N:0 for scoring to show a dominant area (larger is better) of RL-LSFC and RL-FC (RL-LSFCb is very similar to RL-LSFC here and is omitted for viewing). RL-LSFC indeed has a much larger area (difference as in the pure purple area) than RL-FC. (c) Non-dominant column is omitted since all models have 0 in it.

Table 6: Statistics w.r.t. Figure 10, * are our models

	TMDL (L)	AJER (L)	S_{log} (H)	F/S/N
RL-LSFC*	36.84	**88.71**	**2.71**	**50/0/0**
RL-LSFCb*	**14.74**	90.13	2.67	49/1/0
RL-FC	15.28	92.79	2.24	29/21/0
MAF	305.02	92.67	0.65	-

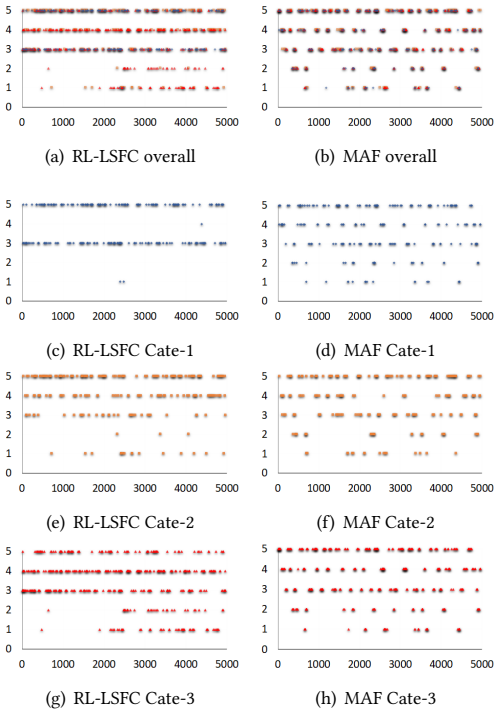
Model behavior pattern exploration

By observing the good performance of RL-LSFC, we would like to further explore interesting behavior patterns of it. We firstly draw temporal job scheduling sequence in one episode (one point for each job, as shown in Figure 11 (a)-(b)) of both RL-LSFC and MAF with respect to all computing clusters, with one color representing one job category. We discover that the deep RL model RL-LSFC indeed shows significantly differentiated scheduling pattern from MAF, which could be coarsely summarized as RL-LSFC tends to utilize clusters with larger capacities and larger heterogeneity factors (thus stronger computing capabilities) more during the early stage of the episode when computing pressure is not peaked, in favor of better performance in both goals. Yet when pressure rises, it well utilizes all cluster resource for overall performance.

Secondly, we further examine the model behavior variance for different job categories (as in Figure 11 (c)-(h)). It seems RL-LSFC can successfully distinguish jobs in different categories and provide differentiated scheduling patterns. Such as, for streaming jobs (Cate-1), it seems to prioritize clusters with stronger computing capability and with elasticity, presumably due to their better potentials in suiting streaming jobs with fluctuated executor amount requests. Such differentiated job categorical scheduling patterns are further illustrated in Figure 12.

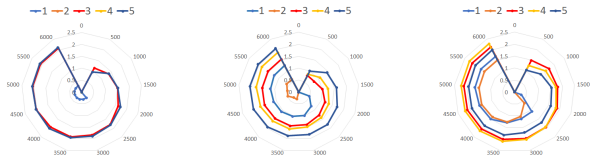
6 CONCLUSION

In this work, we present an elasticity-compatible resource management approach obtained via deep reinforcement learning for a heterogeneous multi-cluster computing environment. In experiment, comparing to the best baseline, it successfully reduces the occurrence of missing execution deadline events for workloads of 1000 jobs by around 5x to 18x in different scenarios, and reduces average execution time ratio by around 2% to 5%; It also shows better performance than a previous reinforcement learning based approach with fully-connected layers. We believe this work can contribute to the progress of utilizing deep reinforcement learning techniques in tackling problems related to large-scale distributed computing environments.



(a) RL-LSFC overall (b) MAF overall
(c) RL-LSFC Cate-1 (d) MAF Cate-1
(e) RL-LSFC Cate-2 (f) MAF Cate-2
(g) RL-LSFC Cate-3 (h) MAF Cate-3

Figure 11: Job-Cluster scheduling patterns for RL-LSFC and MAF in one testing episode. One point for each job and one color for each category. Vertical axis 1-5 are referring to cluster sequence number. Horizontal axis is time slice.



(a) RL-LSFC Cate-1 (b) RL-LSFC Cate-2 (c) RL-LSFC Cate-3

Figure 12: Comparison of Job-Cluster scheduling pattern with respect to different job categories under RL-LSFC control. Value axis is on logarithmic scale of job counts, angle axis is time slice. One color for each cluster.

ACKNOWLEDGMENTS

This work was supported in part by NSF-1836881.

REFERENCES

[1] Enda Barrett, Enda Howley, and Jim Duggan. 2013. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience* 25, 12 (2013), 1656–1674.

[2] Mingxi Cheng, Ji Li, and Shahin Nazarian. 2018. DRL-cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. In *Proceedings of the 23rd Asia and South Pacific Design Automation Conference*. IEEE Press, 129–134.

[3] Xavier Dutreilh, Sergey Kirgizov, Olga Melekhova, Jacques Malenfant, Nicolas Rivierre, and Isis Truck. 2011. Using reinforcement learning for autonomic resource allocation in clouds: towards a fully automated workflow. In *ICAS 2011,*

The Seventh International Conference on Autonomic and Autonomous Systems. 67–74.

[4] Zoltán Gábor, Zsolt Kalmár, and Csaba Szepesvári. 1998. Multi-criteria reinforcement learning.. In *ICML*, Vol. 98. 197–205.

[5] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2016), 2222–2232.

[6] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. 2018. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1–8.

[7] Hatim Khouzaimi, Romain Laroche, and Fabrice Lefèvre. 2016. Reinforcement Learning for Turn-Taking Management in Incremental Spoken Dialogue Systems.. In *IJCAI*. 2831–2837.

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Teng Li, Zhiyuan Xu, Jian Tang, and Yanzhi Wang. 2018. Model-free control for distributed stream data processing using deep reinforcement learning. *Proceedings of the VLDB Endowment* 11, 6 (2018), 705–718.

[10] Ning Liu, Zhe Li, Jielong Xu, Zhiyuan Xu, Sheng Lin, Qinru Qiu, Jian Tang, and Yanzhi Wang. 2017. A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 372–382.

[11] Teng Liu, Xiaosong Hu, Shengbo Eben Li, and Dongpu Cao. 2017. Reinforcement learning optimized look-ahead energy management of a parallel hybrid electric vehicle. *IEEE/ASME Transactions on Mechatronics* 22, 4 (2017), 1497–1507.

[12] Zixia Liu, Hong Zhang, Bingbing Rao, and Liqiang Wang. 2018. A Reinforcement Learning Based Resource Management Approach for Time-critical Workloads in Distributed Computing Environment. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 252–261.

[13] Shie Mannor and Nahum Shimkin. 2004. A geometric approach to multi-criterion reinforcement learning. *Journal of machine learning research* 5, Apr (2004), 325–360.

[14] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 50–56.

[15] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. 2009. Comparison of several cloud computing platforms. In *2009 Second international symposium on information science and engineering*. IEEE, 23–27.

[16] David Silver et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354.

[17] James Supancic III and Deva Ramanan. 2017. Tracking as online decision-making: Learning a policy from streaming videos with reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*. 322–331.

[18] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[19] Kristof Van Moffaert, Madalina M Drugan, and Ann Nowé. 2013. Scalarized multi-objective reinforcement learning: Novel design techniques. In *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 191–199.

[20] Kristof Van Moffaert and Ann Nowé. 2014. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research* 15, 1 (2014), 3483–3512.

[21] Vinod Kumar Vavilapalli et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. ACM, 5.

[22] Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

[23] Jun Wu, Xin Xu, Pengcheng Zhang, and Chunming Liu. 2011. A novel multi-agent reinforcement learning approach for job scheduling in Grid computing. *Future Generation Computer Systems* 27, 5 (2011), 430–439.

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* (2016).