

Received September 26, 2019, accepted October 12, 2019, date of publication October 18, 2019, date of current version October 31, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2948150

# SCARL: Attentive Reinforcement Learning-Based Scheduling in a Multi-Resource Heterogeneous Cluster

MUKOE CHEONG<sup>1</sup>, HYUNSUNG LEE<sup>2</sup>, IKJUN YEOM<sup>1</sup>, AND HONGUK WOO<sup>1</sup> 

<sup>1</sup>Department of Software, Sungkyunkwan University, Suwon 16419, South Korea

<sup>2</sup>Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Honguk Woo (hwoo@skku.edu)

This work was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT and Future Planning under Grant NRF-2016R1E1A1A01943474 and Grant NRF-2018R1D1A1A02086102.

**ABSTRACT** Advanced reinforcement learning (RL) technologies have recently increased the opportunity for automating several tasks in cluster management at scale by exploiting repetitive logs of cluster operation and building a learning model for resource allocation and job scheduling. Yet, this trend of adopting RL in the domain of cluster management has not fully addressed the diversity and heterogeneity of jobs and machines in modern cluster environments. In this paper, we present an RL-based scheduler for a multi-resource cluster, namely SCARL (Scheduler with Attentive Reinforcement Learning), concentrating on intricate cluster operating conditions with different resource requirements and capabilities. Specifically, we employ *attentive embedding* and *factored-action scheduling* that together efficiently incorporate time-varying interdependency of jobs and machines in RL processing; they enable an end-to-end scalable policy for scheduling diverse jobs on heterogeneous machines. To the best of our knowledge, we are the first to employ attention mechanism in RL-based cluster resource management. Through experiments, we demonstrate that our approach is competitive with existing heuristic methods under various cluster simulation configurations, e.g., an average 9.2 % enhancement in slowdown over the shortest job first algorithm. Additionally, the approach yields stable performance with our test cluster for running synthetic workloads based on real traces.

**INDEX TERMS** Cluster resource management, attentive reinforcement learning, attention, attentive embedding.

## I. INTRODUCTION

Resource management and job scheduling in a cluster infrastructure have become important for cost-efficiently operating modern applications of various resource usages with a level of reliability guarantee [1]. Workloads on such a cluster have turned into more diverse, complex, and associated with fast growing data-intensive, machine learning-driven containerized applications. However, current approaches for container scheduling rely on either handcrafted engineering, workload-specific implementation, or static policies [2], [3]. In general, these approaches are restricted in providing flexible abstraction for dealing with different objectives and conditions of multi-resource clusters without reconfiguring underlying system components.

The associate editor coordinating the review of this manuscript and approving it for publication was Shagufta Henna.

The recent advance of reinforcement learning (RL) has expedited the automation of system operations in many areas. They include energy optimization in data centers [4], [5], cluster resource management [6], [7], device placement [8], job placement in cloud networks [9], network slicing [10], [11], and video streaming bitrate optimization [12]. For instance, DeepRM [6] demonstrated the feasibility of using an end-to-end RL structure for cluster resource management. Specifically, the approach represented the cluster operating status as a set of fixed-size state vectors and learned a job scheduling policy on the continual state updates. In this generalized RL structure, rewards, i.e., feedback signals on scheduling decisions, are designed in accordance with a given objective such as reducing the delay of job completion or improving the overall cluster utilization. In that sense, RL-based cluster resource management is considered extensible with such a flexible feedback mechanism.

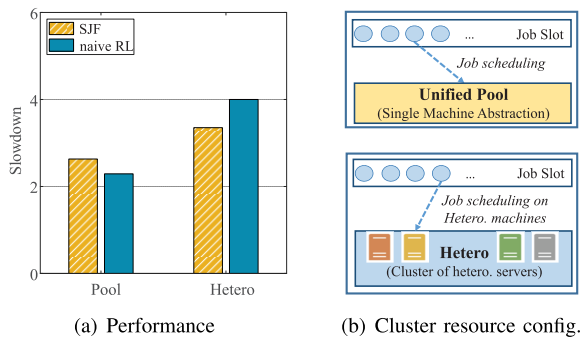


FIGURE 1. Effects of cluster resource configurations.

Specifically, DeepRM presumed a unified resource pool so that it abstracted out the complexity of the underlying cluster operating status and focused on the problem of job selection. Thus, it achieved stable performance in a simulation. Differently from the previous work that leverages resource pool abstraction, in this paper, we concentrate on a multi-resource cluster with heterogeneous machines.

Figure 1 presents one of our initial simulation tests for adopting RL-based job schedulers under different cluster conditions. For comparison purposes, we intentionally implemented an RL model (“naive RL” in the figure) that learns a job selection policy via training, but uses a conventional round-robin algorithm for machine allocation. We specify such an RL model as *naive* because it maintains job features in its fixed-size state vectors with no consideration on embedding. We tested the naive RL model and heuristic algorithms, e.g., shortest job first (SJF), with two cluster simulation configurations such as a unified resource pool (“Pool” in the figure) and a heterogeneous cluster (“Hetero”). As depicted in Figure 1(b), these configurations stipulate from the perspective of job schedulers that the former is seen as a single capable machine while the latter is seen as a set of individual machines with different resource capabilities. Note that more cluster configurations in Table 4 will be discussed in Section V.

As shown in Figure 1(a), the naive RL reveals competitive performance for the unified resource pool configuration, compared to the SJF algorithm. The Y-axis represents overall cluster performance in slowdown, which denotes job completion delay. The slowdown metric will be explained in Equation 6 later. This simulation result is consistent with [6], demonstrating the feasibility of RL-based schedulers in the domain of cluster resource management. For the heterogeneous cluster configuration, however, we observed a less competitive simulation result of the naive RL. This degradation was expected to some extent since the naive RL learns to select jobs without considering the conditions of individual machines. We also tested an end-to-end RL model that learns not only job selection but machine allocation; however, we experienced difficulty in training the model due to its large combinatorial action space.

These simulation results motivate us to investigate RL-based scheduling particularly for a heterogeneous cluster

with intricate operating conditions, and furthermore, to adapt embedding schemes for RL processing. We speculate that such a disparity between limited expressiveness of RL state representation and complex operating conditions of a modern cluster has been turning into a challenge for adopting RL-based scheduling in practice.

To address this challenge, we investigate a scalable embedding technique that is able to deal with the dependency and complexity of time-varying cluster entities. Specifically, we consider these two characteristics as important interdependent factors in scheduling: (1) workload diversity with various resource requirements for jobs, and (2) cluster heterogeneity with different machine resource capabilities that are common in current cluster operating environments. Note that (1) and (2) were studied individually in RL-based scheduling (e.g., [6], [7] for (1); [13], [14] for (2)). However, their combined implication was not thoroughly investigated. For incorporating RL models into cluster resource management with the aforementioned characteristics, we formulate the scheduling decision as sequential job selection and machine allocation, and solve the following problems.

First, containerized applications and virtualized cloud infrastructures allow us to deploy and operate multi-tenant applications cost-efficiently through standard, high-level interfaces; these increase the complexity and heterogeneity of cluster workloads and systems in terms of resource management [18]–[21]. Emergent technology trends such as Internet of Things (IoT), edge computing, and network function virtualization (NFV) further increase the complexity by incorporating a new type of devices into distributed cluster environments [22], [23]. We employ attentive embedding for representing such complex properties of cluster entities, and implement a layered attention mechanism for capturing the bi-directional impact of entity pairs within this hierarchy. We consider jobs and machines as cluster entities of which the representation should be combined to provide a better understanding on overall cluster operating status. Our approach follows the recent work that extends graph embedding for complex job representation [7]. However, our interest is not on the internal structure of individual jobs but on the interdependency of jobs and machines for a multi-resource heterogeneous cluster.

Second, it is desirable to have rich expressiveness on the cluster operating status when adopting RL-based schedulers. Furthermore, it is also important to maintain scalable representation of RL states and actions to achieve a robust learning process. To this end, combining with attentive embedding, we explore factored-action scheduling that transforms combinatorial actions into sequential actions and facilitates the learning process with an action set of manageable size.

The main contributions of this paper are summarized as follows:

- We propose an RL-based job scheduler that exploits attention mechanism, namely SCARL (Scheduler with Attentive Reinforcement Learning), considering wide diversity and heterogeneity imposed on jobs and

**TABLE 1.** Summary of related studies.

|                                | Cluster config.        | Workload config.                                                                              | Method                                                                                     | Objective                                                                          |
|--------------------------------|------------------------|-----------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------|
| <i>Paragon</i> [15]            | Heterogeneous machines | Various tasks, e.g., multi-threaded; multi-programmed; I/O bound                              | Collaboration filtering based classification for unknown apps, and greedy server selection | QoS, e.g., maintaining QoS for 91% and guaranteeing 52% of workloads               |
| <i>Quasar</i> [16]             | Heterogeneous machines | Different service scenarios with perf. constraints, e.g., low-latency; single- or multi-batch | Collaboration filtering based classification for resource allocation, type, interference   | Cluster utilization improvement than reservation (while meeting perf. constraints) |
| <i>SmartYARN</i> [17]          | Unified pool           | Cloud and MapReduce jobs with service level agreement                                         | RL - Q-learning alg. with perf. and cost based rewards                                     | Resource usage optimization for long-running time apps                             |
| <i>DeepRM</i> [6]              | Unified pool           | Multi-resource (e.g., CPU, mem.) independent, non-preemptive jobs                             | Deep RL - REINFORCE alg. variant (policy gradient method)                                  | Reduction in avg. slowdown or job completion time                                  |
| <i>Decima</i> [7]              | Homogeneous machines   | Data processing jobs with dependent stages, e.g., Spark jobs                                  | Deep RL framework with graph neural network and policy network                             | Reduction in avg. job completion time                                              |
| <i>DRF</i> variants [18], [19] | Heterogeneous cluster  | Multi-resource Spark/Hadoop jobs                                                              | Generalized DRF with best-fit in [18], XGBoost model in [19]                               | Increase fairness and utilization, reduction in job completion time                |
| <i>Tetris</i> [20]             | Data-parallel cluster  | Multi-stage, MapReduce jobs with multi-resource requirements                                  | Multi-dimensional bin packing with dot product on euclidean space                          | Reduction in avg. makespan, avg. job completion time                               |

machines in a cluster. To do so, we devise the attentive embedding that learns to capture time-varying interdependent features of jobs and machines, and renders the features confined to concise yet scalable state representation.

- We address the scale issue of RL actions by employing the sequential scheduling policy based on factored actions.
- We demonstrate that our approach is competitive with well-known heuristic scheduling algorithms under heterogeneous cluster simulation settings, e.g., 9.2 % improvement on average slowdown. Furthermore, we observe stable performance of SCARL with our Kubernetes cluster for running synthetic workloads reproduced from real traces.

The rest of the paper is organised as follows. Section II reviews learning-based cluster resource management and job scheduling. Section III presents the proposed SCARL architecture including cluster environment, RL-based scheduler, and attentive representation, and then Section IV describes the attentive RL method for scheduling jobs on a multi-resource heterogeneous cluster. Finally, Section V provides the experiment results and Section VI concludes the study.

## II. RELATED WORK

There have been several research studies and industry projects for cluster resource management such as Borg [1], YARN [24], and Kubernetes [25] which commonly support cost-efficient and reliable operation of diverse workloads on a general-purpose computing infrastructure.

With the growing popularity of machine learning (ML) techniques, researchers have used cluster resource management as an area where ML-based system optimization is productive. The related studies in this literature are summarized in Table 1. For example, [15], [16] employed collaborative filtering using matrix factorization for matching jobs and machines in that a matched pair maps to a scheduling decision on a cluster. In particular, a few works recently exploited

reinforcement learning. In [17], SmartYarn extended Apache YARN by adopting a deep Q-network which can reduce the resource usage for meeting service level agreements of a cloud service. In [6], DeepRM showed how to adopt deep neural networks and policy gradient-based learning for cluster resource management, presuming the simplified assumption on server resources as a unified pool. In [7], the Decima RL-based system aimed at dealing with data-intensive complex workloads on a cluster, and investigated the graph embedding for extracting features of complicated data processing jobs and translating them in a vector representation. Similarly, in [26], an RL-based solution was implemented on top of YARN for scheduling batch analytics workloads. As such, these works normally addressed the complexity of jobs, but rarely stipulated heterogeneous machine capabilities.

Recently there have been a few research works focusing on the problem of fair resource allocation on a cluster with server heterogeneity. DRFH in [18] generalized the notion of dominant resource fairness (DRF) [27] by adapting the max-min fairness and exploiting a best-fit heuristic scheme for resource allocation on multiple heterogeneous servers. X-DRF [19], a learning-based resource allocation using the XGBoost ML algorithm was introduced in the same problem context. As opposed to the above variants of DRF, in [20], the authors explored heuristic methods for the multi-dimensional bin packing problem and adapted them to the context of job scheduling with diverse resource demands, showing that the packing-based heuristics are compatible with fairness policies (i.e., DRF). These previous studies considered multi-resource heterogeneous servers as a feature of cluster infrastructures. However, they adopted heuristics-based or conventional machine learning schemes rather than modern deep learning or advanced reinforcement learning-based scheduling schemes.

We notice that both jobs and machines influence the operation performance, especially when they are not identical in terms of resource requirements and capabilities. Accordingly, when adapting RL-based scheduling to a multi-resource heterogeneous cluster, we regard the interdependency of

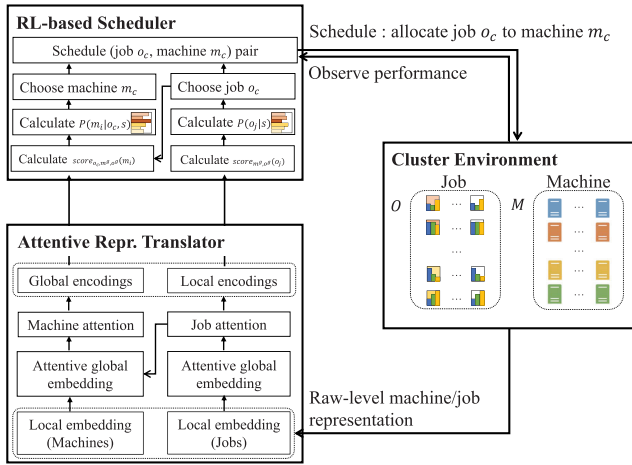


FIGURE 2. Overall system design.

jobs and machines as an important feature, and investigate scalable embedding techniques including attention mechanism to incorporate the interdependency in RL state representation.

The concept of attention was introduced in the context of deep learning architectures for machine translation [28], [29], and its application areas have been rapidly expanded to others such as recommendation [30], image generation [31], image recognition [32]–[34]. In [35], the relational reasoning architecture showed how to map an entity to local region of an image (e.g., game units in Starcraft II) by using the attention-based embedding. A couple of works recently adopted attention-based RL techniques for solving combinatorial optimization problems such as the travel salesman problem [36], [37].

In this paper, we take a similar step to adopt the attention-based embedding for representing the RL state at scale, but we further exploit the apriori known relation of jobs and machines, i.e., whether a job is *allocable* on a machine (by Equation 3). As a result, combinatorial actions are transformed into sequential actions, expediting the iterative embedding steps in learning the RL policy. To the best of our knowledge, we are the first to employ attention mechanism for RL-based cluster resource management.

### III. OVERALL SYSTEM

In this section, we describe our system design exploiting deep RL techniques in the context of multi-resource heterogeneous cluster management. As abstracted in Figure 2, the system design includes (1) cluster environment, (2) RL-based scheduler, and (3) attentive representation translator. We first explain our scheduling problem with the cluster environment, and then describe the problem formulation in RL structure.

#### A. CLUSTER ENVIRONMENT

We first consider a set of cluster machines and a set of container-based jobs (or jobs):

$$M = \{m_1, m_2, \dots, m_{|M|}\}, \quad O = \{o_1, o_2, \dots, o_{|O|}\}.$$

where  $|M|$  and  $|O|$  denote the numbers of machines and jobs respectively. We assume that  $M$  and  $O$  can change over time, and their representation is associated only with the operating machines and outstanding jobs that are currently observed in a cluster environment.

We then consider  $d_r$ -different resource types, e.g., CPU, memory, I/O, GPU, and network. Given such multiple resource types, each machine  $m_i \in M$  is represented by two  $d_r$ -dimensional vectors and the processing factor:

$$((rc_{i1}, \dots, rc_{id_r}), (ra_{i1}, \dots, ra_{id_r}), pfi). \quad (1)$$

For each resource type  $k \in \{1, \dots, d_r\}$ ,  $ra_{ik}$  denotes the currently available resource capability of the machine  $m_i$  that can be allocated for a new job, and  $rc_{ik}$  denotes the total resource capability of  $m_i$  by combining  $ra_{ik}$  and the resource capability already allocated for the jobs running on  $m_i$ . And  $pfi \in (0, 1]$  is a normalized processing factor that represents the computational machine capability such that a low  $pfi$  indicates a job proportionally requires more execution time on  $m_i$ .

Similarly each job  $o_j \in O$  is represented by a pair of a  $d_r$ -dimensional vector for its resource requirement and a scalar value for its computation time:

$$((rr_{j1}, \dots, rr_{jd_r}), ct_j). \quad (2)$$

In addition, the deadline  $dl_j$  can be added in this job representation for real-time tasks. In that case,  $dl_j$  specifies that  $o_j$  should be completed within  $dl_j$ ; otherwise, it is violated.

Given the representation of machines and jobs above, a job  $o_j \in O$  is *allocable* on a machine  $m_i \in M$  only if

$$ra_{ik} - rr_{jk} \geq 0, \quad \forall k \in \{1, \dots, d_r\} \quad (3)$$

holds. If there exists a machine  $m_i \in M$  such that Equation 3 holds for  $o_j$ , we say that  $o_j$  is *allocable* in the cluster. For simplicity, we use the normalized resource representation such that

$$\sum_{i=1}^{|M|} rc_{ik} = 1 \quad \forall k \in \{1, \dots, d_r\} \quad (4)$$

holds.

Here, we make several assumptions consistent with previous works in cluster resource management [6], [16]. (i) We first consider a long-running cluster environment where the majority of jobs are repetitive, and thus the computation time bound  $ct_j$  can be identified. In this setting, it is presumed that each job arrives at the cluster either online or in batch, and its features (in Equation 2) are immediately known upon arrival. A scheduler is then invoked to make decisions about a subset of  $O$  being allocated upon any event relevant to changes in a cluster operating environment, e.g., job completion or arrival. (ii) We only consider non-preemptive jobs. Thus a cluster operates in such a way that for a job  $o_j$  being allocated on a machine  $m_i$ , the allocated resources of  $m_i$  are released after  $(ct_j \times \frac{1}{pfi})$  time and then become allocable for another jobs.

Considering modern cluster operating environments with containerized jobs, in addition, we are interested in highly dynamic cluster operating conditions such that there might be a wide variance of resource requirements across jobs and their arrival patterns, and the cluster size  $|M|$  might change over time, e.g., the cluster dynamically scales in and out. Such dynamic conditions have not been intensively investigated in previous works.

Regarding the objective of operating a cluster, we consider two specific performance metrics, slowdown and job completion, while the generic RL structure of SCARL allows us to take into consideration more than these metrics. Having a stream of job arrival events, the scheduler intends to exploit the RL policy learned to continuously make appropriate decisions on selecting a subset of outstanding jobs and allocating one of the available machines for each selected job. In general, such an RL policy can be tailored by the reward strategy toward optimizing a given objective while the common RL structure is maintained independent of the objectives.

## B. RL FORMULATION FOR SCHEDULING

Here we present the RL formulation on the cluster entities described above, on which an RL-based scheduler can be designed. In general, an RL problem is structured with several components such as *agent*, *state*, *action*, *reward*, and *environment* where an agent continuously interacts with an environment in which the operating status is represented by a state. The interaction includes actions from the agent to the environment as well as reward feedback and state updates from the environment to the agent. In the following, we provide the RL formulation for the state, action, and reward shared by the cluster (environment) and scheduler (agent) in the context of our problem.

*State*: The state of the cluster operating status is represented by jobs and machines:

$$s = (O : (o_1, o_2, \dots, o_{|O|}), M : (m_1, m_2, \dots, m_{|M|})). \quad (5)$$

*Action*: Given a set of jobs  $O$  and machines  $M$ , it is noticed that the whole space of scheduling actions in the RL formulation can become as large as  $\sum_{i=1}^{|O|} \binom{|O|}{i} \times |M|^i$ . This implies that learning to schedule jobs in a multi-resource heterogeneous cluster should address such a challenge of large combinatorial action spaces.

To facilitate the learning process, we instead employ sequentially factored actions in RL. With an individual action for selecting a job from a set of outstanding jobs as well as a machine from the available cluster machines to place the job, it is possible to reduce the action space to  $\mathcal{O}(|M| \times |O|)$  or  $\mathcal{O}(|M| + |O|)$  through a sequence of such factored actions. For former case, we use combinatorial scheduling for a job and machine pair whereas for latter case, we use sequential scheduling for each pair. Both commonly produce a set of machine and job pairs.

In doing so, we rely on the time semantics tailored for RL-based scheduling in which the *timestep* of RL processing

and the actual (system) time of job processing on the cluster machines are not the same [6]. The RL timestep keeps monotonically increasing when a set of jobs are sequentially selected by a scheduler, but the actual time does not. The actual time flows only if none of the outstanding jobs are chosen by a scheduler. This time semantics is consistent from the perspective of simulating cluster operation since the jobs being sequentially selected until no job is allocable (i.e., when the actual time flows again) correspond to a set of jobs that simultaneously begin running on the cluster.

*Reward*: In principle, learning to schedule in the RL structure requires a properly designed reward strategy for a target objective. Aiming at optimizing the performance of a cluster, we use the slowdown metric for a job  $o_j$  similarly to [6] and accordingly use the following reward strategy

$$\text{slowdown} = \frac{ct'_j}{ct_j}, \quad \text{reward} = -\tau \sum_{j \in O^+} \frac{\epsilon}{ct_j} \quad (6)$$

where  $ct_j$  is the computation time of  $o_j$  and  $ct'_j$  is the response time of  $o_j$  meaning the total elapse time from  $o_j$ 's arrival to completion.  $O^+$  denotes a set of jobs arrived and yet to be finished including the outstanding jobs in  $O$ ,  $\tau$  denotes the elapse time from the previous timestep, and  $\epsilon$  denotes the configurable penalty parameter.

In addition, for the generalized real-time task model where the utility of a job becomes zero if its deadline is violated, we use the job completion rate as the objective metric and accordingly set  $\text{reward} = -|V|\epsilon$  where  $V$  denotes a set of jobs of which the deadline constraint is being violated since the previous timestep.

## C. ATTENTIVE REPRESENTATION

For representing the operating status of a cluster, we adopt the attention mechanism introduced in [28] and [29]. This attention mechanism has become popular in training neural networks particularly for the cases when the relation of inter-dependent features needs to be dynamically analyzed.

Upon a scheduling event during cluster operation, as shown in Figure 2, the cluster environment provides the RL-based scheduler with the raw-level data as input for RL processing. In our proposed system architecture, the raw-level data is transformed into the scalable system state via the embedding process based on the attention mechanism. Recall that our interest is on such a dynamic condition where heterogeneous machines and diverse jobs of variable-size are continuously observed during cluster operation. We employ attention-based embedding networks to represent the dynamic condition efficiently. Next, we provide the formal definition of attention, followed by a detailed explanation.

In general, the attention function below computes the weighted sum that indicates how relevant (important) the query vector  $q$  is for a sequence of vectors  $X = [x_1, \dots, x_N]$

$$\text{att}(q, X) = \sum_{i=1}^N \alpha_i x_i \quad (7)$$

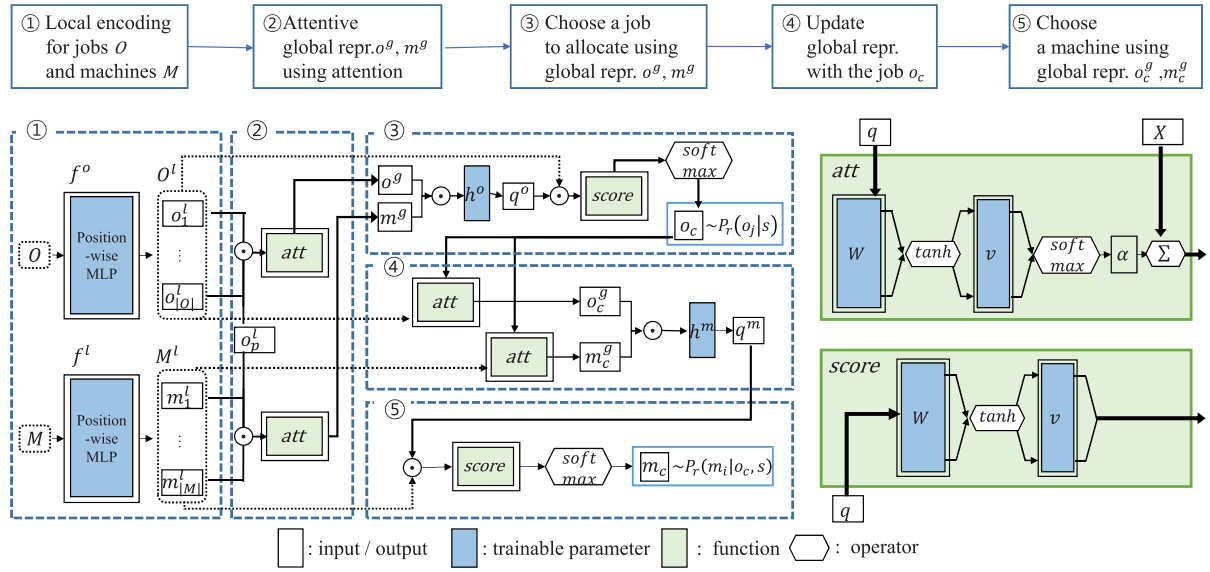


FIGURE 3. Attentive RL structure.

where  $\alpha_i$  is the normalized alignment score using the softmax function  $\alpha_i = \frac{\exp(u_i)}{\sum_{i'=1}^N \exp(u_{i'})}$  and  $u_i$  is the alignment score of  $x_i \in X$  with respect to the query  $q$ :

$$u_i = v^T \tanh W [x_i; q].$$

Note that  $[x_i; q]$  represents concatenation of vectors. Trainable parameters  $v$  and  $W$  are used throughout the following sections.

#### IV. ATTENTIVE REINFORCEMENT LEARNING

In this section, we describe how to adopt an attention mechanism in our proposed RL-based scheduler and mitigate the scheduling complexity of dynamic cluster entities as well as multiple resource types in a heterogeneous cluster. Figure 3 illustrates our attentive RL structure that represents the cluster operating status efficiently, thereby being likely to render the learned model more steadily and improve the target objective performance. Table 2 contains the symbol notations used throughout the following sections.

##### A. ATTENTIVE EMBEDDING

We incorporate multiple learning layers in embedding process for representing the cluster operation status lightly. Given a sequence of feature vectors of jobs  $O = [o_1, o_2, \dots, o_{|O|}]$ , SCARL transforms it to the local encoding via a trainable nonlinear function  $f^o(\cdot)$ . The same is applied for machines, so we have

$$\begin{aligned} O^l &= [f^o(o_1), f^o(o_2), \dots, f^o(o_{|O|})], \\ M^l &= [f^m(m_1), f^m(m_2), \dots, f^m(m_{|M|})]. \end{aligned} \quad (8)$$

Subsequently SCARL aggregates a set of local per-job (or per-machine) encoding vectors into a single vector to represent a summary of all the job (or machine) states by exploiting the attention mechanism. The summarized global

TABLE 2. Notations.

| Symbol         | Description                                                            |
|----------------|------------------------------------------------------------------------|
| $s$            | cluster state $s = (\{o_1, \dots, o_{ O }\}, \{m_1, \dots, m_{ M }\})$ |
| $o_j^l, m_i^l$ | local encoding of job $o_j$ and machine $m_i$                          |
| $O^l, M^l$     | matrix of all local job and machine encodings                          |
| $o_c, m_c$     | job and machine chosen currently                                       |
| $o_p^l$        | local encoding of job chosen at the previous timestep                  |
| $o^g, m^g$     | global encoding of all outstanding jobs and machines                   |
| $o_c^g, m_c^g$ | global encoding corresponding to job chosen currently                  |
| $f^o, f^m$     | nonlinear functions for local encoding                                 |
| $h^o, h^m$     | nonlinear functions for abstracting cluster operating status           |
| $q^o, q^m$     | query repr. for cluster operating status                               |

representation of all the outstanding jobs and the available machines are calculated as

$$o^g = \text{att}^o(o_p^l, O^l), \quad m^g = \text{att}^m(o_p^l, M^l) \quad (9)$$

where  $o_p^l$  is the local encoding of the job selected at the previous timestep. This embedding process is depicted in the left boxes numbered as 1 and 2 in Figure 3.

##### B. SCHEDULER BASED ON FACTORED ACTIONS

SCARL leverages the complexity reduction by transforming a combinatorial scheduling decision about interdependent jobs and machines to a factored action pair in the RL structure. This enables RL models to learn online scheduling policies along with massive state changes. The middle boxes numbered as 3, 4, and 5 in Figure 3 illustrate how factored actions are made over iterative attentive embedding.

In general, an RL policy  $\pi(a, s)$  is defined as the probability distribution that an action  $a$  is taken at a given state  $s$ . Suppose we have a pair of job and machine  $(o_c, m_c)$  (by Equation 11~14 below); then, the factored actions for scheduling the pair, i.e., selecting the job  $o_c \in O$  and allocating the

machine  $m_c \in M$  for  $o_c$ , can be seen as such a policy that:

$$\pi((o_c, m_c), s) = \Pr(o_c, m_c | s) = \Pr(o_c | s) \Pr(m_c | o_c, s).$$

In the following section, we explain how to acquire the distributions through RL processing. Recall that the size of job set  $O$  and machine set  $M$  might be time-varying; the number of pending jobs changes by job arrivals and scheduling decisions, and the number of operating machines also changes by scaling or failure events in a cluster. Rather than confining such a variable-size set restrictively on static action representation, we employ the notion of Pointer [38] to dynamically compute the probability of a job and machine pair from the variable-size job and machine sets.

To do so, we first calculate the score of each job with respect to the input query  $q^o = h^o([m^g; o^g])$  (which contains the attentive global representation regarding the machines and the jobs in the cluster) for a trainable nonlinear function  $h^o(\cdot)$  by

$$\text{score}_{m^g, o^g}(o_j) = \begin{cases} C v^T \tanh W[o_j; q^o] & o_j \text{ is allocable} \\ -\infty & \text{otherwise} \end{cases} \quad (10)$$

where  $C$  is the clipping constant (default = 5) such that clipping  $\text{score}_{m^g, o^g}(o_j) \in [-C, C]$ .

Note that there might be several jobs that cannot be allocated due to the disparity of resource availability and resource requirement; those jobs are given an  $-\infty$  score for pruning. Having the scores of all allocable jobs, then, we interpret those as the probability distribution of job selection,  $\Pr(o_j | s)$ , via softmax normalization

$$\Pr(o_j | s) = \frac{\exp(\text{score}_{m^g, o^g}(o_j))}{\sum_{j'} \exp(\text{score}_{m^g, o^g}(o_{j'}))} \quad (11)$$

where  $s$  denotes the state representing the cluster operating status (in Equation 5). The probability distribution is learned through the attentive global representation of cluster entities, i.e.,  $m^g$  and  $o^g$ , that efficiently captures the relevant interdependency in  $s$ . Given the sampled job  $o_c$  (the job chosen currently) based on the distribution above, the global representation of jobs and machines is updated by:

$$o^g = \text{att}^o(o_c^l, O^l), \quad m^g = \text{att}^m(o_c^l, M^l). \quad (12)$$

Subsequently, for machine allocation, the score of machines with respect to the job  $o_c$  is calculated by

$$\text{score}_{o_c, m^g, o_c^g}(m_i) = \begin{cases} C v^T \tanh W[m_i; q^m] & o_c \text{ is allocable on } m_i \\ -\infty & \text{otherwise} \end{cases} \quad (13)$$

where  $C$  is the clipping constant,  $q^m = h^m([m^g; o^g; o_c])$  and  $h^m(\cdot)$  is a trainable nonlinear function. Based on the distribution below, a machine for running the job  $o_c$  is chosen.

$$\Pr(m_i | o_c, s) = \frac{\exp(\text{score}_{o_c, m^g, o_c^g}(m_i))}{\sum_{i'} \exp(\text{score}_{o_c, m^g, o_c^g}(m_{i'}))} \quad (14)$$

### Algorithm 1 Attentive Reinforcement Learning

*// 1. Local encoding for jobs and machines*

$$O^l = [o^l(o_1), \dots], M^l = [m^l(m_1), \dots] \quad // \text{eq.}(8)$$

*// 2. Attentive global repr. using attention mechanism*

$$\text{Compute } o^g = \text{att}^o(o_p^l, O^l), m^g = \text{att}^m(o_p^l, M^l) \quad // \text{eq.}(9)$$

*// 3. Choose a job to allocate using global repr.  $o^g, m^g$*

$$\text{Compute } \text{score}_{m^g, o^g}(o_j) \quad // \text{eq.}(10)$$

$$\text{Compute } \Pr(o_j | s) = \frac{\exp(\text{score}_{m^g, o^g}(o_j))}{\sum_{j'} \exp(\text{score}_{m^g, o^g}(o_{j'}))} \quad // \text{eq.}(11)$$

Sample job  $o_c$  from  $\Pr(o_j | s)$

*// 4. Update global repr. with the job  $o_c$*

$$\text{Compute } o^g = \text{att}^o(o_c^l, O^l), m^g = \text{att}^m(o_c^l, M^l) \quad // \text{eq.}(12)$$

*// 5. Choose a machine using global repr.  $o^g, m^g$*

$$\text{Compute } \text{score}_{o_c, m^g, o_c^g}(m_i) \quad // \text{eq.}(13)$$

$$\text{Compute } \Pr(m_i | o_c, s) = \frac{\exp(\text{score}_{o_c, m^g, o_c^g}(m_i))}{\sum_{i'} \exp(\text{score}_{o_c, m^g, o_c^g}(m_{i'}))} \quad // \text{eq.}(14)$$

Sample machine  $m_c$  from  $\Pr(m_i | o_c, s)$

*// Schedule the pair of job and machine*

Allocate machine  $m_c$  for job  $o_c$

The steps explained above are all compiled in Algorithm 1 where the comments are numbered as in Figure 3.

### C. TRAINING RL

For training RL models, we use the policy-gradient method which is known to be effective for end-to-end learning on neural networks at scale [39]. The RL processing based on the policy-gradient method works as follows. In the RL structure, at each timestep  $t$ , an agent observes the state  $s_t$  of an environment and then it chooses an action  $a_t$  based on the policy  $\pi_\theta$  parameterized by  $\theta$ . Conducting  $a_t$ , the environment changes in that the state moves from  $s_t$  to  $s_{t+1}$ . Subsequently the agent receives a reward  $r_t$ .

In this processing, the policy can be updated using the experience including the stream of state, action, and reward samples. This learning process is to maximize the objective function  $J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$  where  $\gamma \in (0, 1]$  is a discount factor that discounts future rewards.

We use the REINFORCE algorithm [40] to train RL models. Given a series of simulation episodes along with the policy  $\pi_\theta$  recorded as  $(s_0, a_0, r_0, s_1, \dots, s_T, a_T, r_T)$ , the parameters  $\theta$  are gradually optimized by the policy gradient method formalized as

$$\theta \leftarrow \theta + \lambda \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) (G_t - b(s)) \quad (15)$$

where the return  $G_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$  is the cumulative rewards from time  $t$  to termination  $T$ , and  $\lambda$  is the step size configurable through experiments [39]. To reduce the variance of the gradient estimates from each return  $G_t$ , we use the baseline function  $b(\cdot)$ . Specifically, similar to [41], we exploit the input-dependent baseline  $b_t = \frac{1}{N} \sum G_t$  by which the variances caused by different job sequences are likely to diminish. The RL processing by Equation 15 normally follows the updating scheme such that it increases  $\pi_\theta(s_t, a_t)$  if the return  $G_t > b(s)$  holds, and decreases, otherwise. The updating amount is proportional to  $|G_t - b(s)|$ .

**TABLE 3. Cluster entity configuration: Job and machine parameters.**

| Entity            | Parameter                         | Type  | Desc. ( $\forall k, k' = 1, \dots, d_r$ )                            |
|-------------------|-----------------------------------|-------|----------------------------------------------------------------------|
| Job ( $o_j$ )     | Resource requirement variance     | Zero  | $rr_{jk} = rr_{jk'}$                                                 |
|                   |                                   | High  | $\delta_l \leq \frac{\max(rr_{jk})}{\min(rr_{jk'})} \leq \delta_u^*$ |
|                   |                                   | Low   | $\frac{\max(rr_{jk})}{\min(rr_{jk'})} < \delta_l$                    |
|                   | Computation time                  | Long  | $ct_j \in (40, 100)$                                                 |
|                   |                                   | Short | $ct_j \in (1, 30)$                                                   |
|                   | Deadline                          | Exist | $dt_j \in (1, 500)$                                                  |
|                   |                                   | None  | $dt_j = \infty$                                                      |
| Machine ( $m_i$ ) | Resource capability variance (CV) | Zero  | $rc_{jk} = rc_{jk'}$                                                 |
|                   |                                   | High  | $\delta_l \leq \frac{\max(rc_{jk})}{\min(rc_{jk'})} \leq \delta_u$   |
|                   |                                   | Low   | $\frac{\max(rc_{jk})}{\min(rc_{jk'})} < \delta_l$                    |
|                   | Processing ability (PA)           | Fast  | $pf_i = 1$                                                           |
|                   |                                   | Slow  | $pf_i \in \{0.3, 0.4, 0.5\}$                                         |

\* Default :  $\delta_l = 2, \delta_u = 3$

## V. EVALUATION

In this section, we evaluate the performance and resilience of SCARL under various experiment settings, comparing with several heuristic methods. We consider two performance objectives: (1) maximizing the cluster performance in terms of job completion delay (i.e., slowdown) and (2) supporting jobs with deadline constraints. We implement a simulator for a multi-resource heterogeneous cluster environment, which can interact with a job scheduler. We exploit a set of cluster and workload configurations intentionally designed for verifying the behavior of SCARL through simulation. We also make use of real traces from Google data centers [21], [42], [43] to emulate workloads running on our test cluster managed by Kubernetes.

### A. SIMULATION SETTING

Here we provide our simulation configurations for a multi-resource heterogeneous cluster with diverse workloads.

Table 3 lists simulation parameters by which jobs and machines can be generated with specific resource characteristics for experiments. The job parameters including resource requirement variance, completion time, and deadline confine the range of raw feature values of each job as well as specify the diversity of jobs. The machine parameters do the same for generated machines. Given raw feature values of jobs and machines (in Equation 1 and 2), we use respective normalized values (in Equation 4) so that they can be transformed into vector or matrix representation for multilayer perceptron (MLP) processing.

Table 4 describes the heterogeneity levels of a cluster. The Lv1 option is set to have a cluster of homogeneous machines sharing the same property, Lv2 is set to have a cluster of different machines with low variances in capabilities (low heterogeneity), and Lv3 is set to have a cluster of different machines with high variances in capabilities (high heterogeneity) and processing factor. We also include the option of

**TABLE 4. Cluster configuration: Heterogeneity parameters.**

| Level | Description           | $ M ^a$ | $CV^b$              | $PA^c$               |
|-------|-----------------------|---------|---------------------|----------------------|
| Lv0   | Unified resource pool | 1       | -                   | -                    |
| Lv1   | Homogeneous machines  | [3..10] | Zero:100%           | Fast:100%            |
| Lv2   | Low heterogeneity     | [3..10] | Low:100%            | Fast:100%            |
| Lv3   | High heterogeneity    | [3..10] | High:70%<br>Low:30% | Fast:80%<br>Slow:20% |

<sup>a</sup> The number of machines; default:  $|M| = 5$

<sup>b</sup> Resource capability variance of machines in Table 3

<sup>c</sup> Processing ability of machines in Table 3

**TABLE 5. Hyperparameter setting.**

|                 | Description                | Default Setting            |
|-----------------|----------------------------|----------------------------|
| Episode Setting | Num. of epochs             | 500                        |
|                 | Num. of job seq. per epoch | 4                          |
|                 | Num. of training job seq.  | 1000                       |
|                 | Num. of test job seq.      | 200                        |
| Network Setting | Nonlinear func.            | MLP/1 hidden layer/size 32 |
|                 | Attention Mechanism        | Bahanadu Attention/size 32 |
| Model Fitting   | Learning rate              | 0.001                      |
|                 | Optimizer                  | RMSProp                    |

a unified resource pool (Lv0), that is a single capable machine and is defined only for comparison purposes.

Our RL implementation is based on Python 3.6 and PyTorch 1.0. For encoding the raw data input of each job and machine, we feed their raw feature vectors to the position-wise MLP with one hidden layer with a size of 32 that produces 32-dimensional vectors (i.e., local embedding). Note that we set all local and global encodings to be 32-dimensional vectors for computational efficiency.

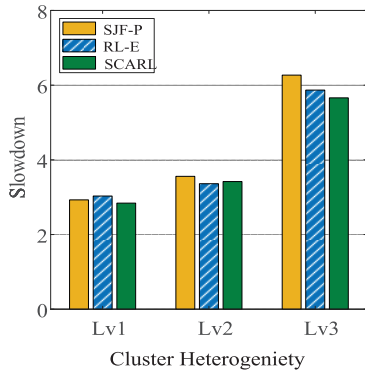
Table 5 lists the hyperparameter settings for model training. For each experiment, we train the model with randomly generated 1000 job sequences (job seq. in the table) and 500 epochs. A job sequence corresponds to a time-series of job samples in simulation workloads, where the job arrival rate follows a Poisson distribution with mean  $\in [2, 4]$ . For each epoch, 4 job sequences are randomly chosen from the training job sequences. Each selected sequence runs 20 times to calculate its baseline, and therefore, 80 sequences are executed during an epoch. Note that it takes 70 seconds on average for completing an epoch on our machines with Intel(R) Core(TM) i7-8700 CPU. We use the RMSProp optimizer to train a model because it shows stable performance in all experiments, and the learning rate sets to 0.001.

### B. RL MODELS

For evaluation, we implemented RL models as well as several heuristic methods including SJF (Shortest Job First), LST (Least Slack Time), and EDF (Earliest Deadline First) for job selection, and best fit packing (similar to Tetris [20]) for machine allocation. Our proposed RL model in SCARL employs attention for efficiently driving the global representation of the cluster operating status and learning to extract the interdependency of jobs and machines.

**TABLE 6. Models in comparison.**

| Method   | Job selection                 | Machine allocation |
|----------|-------------------------------|--------------------|
| SJF-P    | Shortest Job First            | Best Fit Packing   |
| LST-P    | Least Slack Time              | Best Fit Packing   |
| EDF-P    | Earliest Deadline First       | Best Fit Packing   |
| RL model |                               |                    |
| RL-E     | Model description             |                    |
| SCARL    | RL with Global representation |                    |
|          | RL with Attentive embedding   |                    |

**FIGURE 4. Cluster configuration: Heterogeneity.**

A summary of the RL models and heuristic methods is in Table 6. The RL-E model is a variant of SCARL, which works similarly except that it simplifies the embedding process. RL-E uses both local-level and global-level encoded data, but performs a non-attentive aggregation for global representation of cluster operating status, e.g.,  $o^g = \frac{1}{|O|} \sum_j o_j^l$ ,  $m^g = \frac{1}{|M|} \sum_i m_i^l$ .

### C. PERFORMANCE

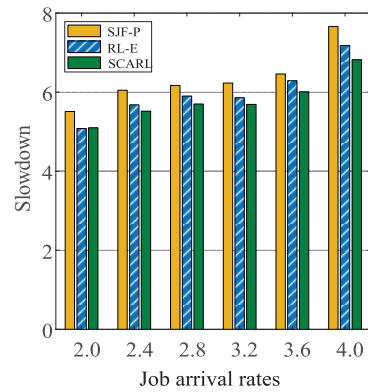
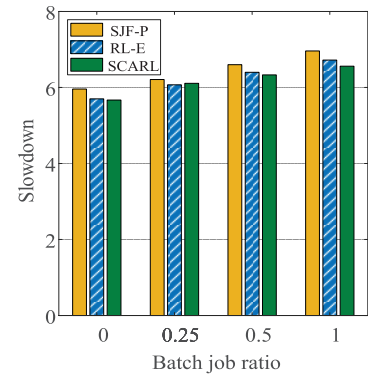
Here, we describe the experiments in our cluster simulation environment and discuss the performance of SCARL under different cluster conditions.

#### 1) HETEROGENEOUS CLUSTER TEST

We first evaluate the cluster performance with respect to the heterogeneity, considering the characteristics of a cluster in terms of the difference among cluster machine members in a group, as described in Table 4. Figure 4 shows the job slowdown with respect to the different levels of cluster heterogeneity (on X-axis).

At high heterogeneity (Lv3), SCARL outperforms the other methods, showing 10.7 % improvement over SJF-P. RL-E also reveals better performance than SJF-P by 6.8 %. This result demonstrates the benefit of attentive embedding and global representation on the cluster operating status, especially when complex conditions are given. SCARL is able to learn to extract the interdependent features of jobs and machines, hence rendering its scheduling decision more stable.

However, at low heterogeneity (Lv2), the performance gain of SCARL over SJF-P reduces to 4.1 %. At Lv1 where a cluster of homogeneous machines is given, the effect of

**FIGURE 5. Workload configuration: Job arrival rates.****FIGURE 6. Workload configuration: Batch job ratio.**

embedding seems to be unpredictable and the performance gain diminished. We observe that the machines with the same resource capabilities frequently look similar in terms of their local encoded representation, and thus the advantage of using the attentive global representation becomes less significant.

In the following, we set our cluster simulation environment to the Lv3 heterogeneity, unless stated otherwise, for concentrating on our target scenarios.

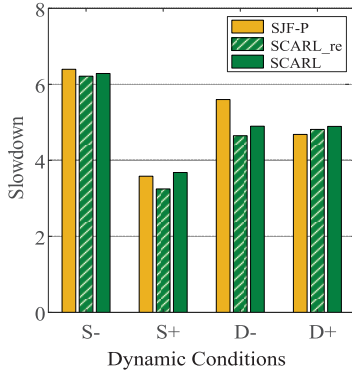
#### 2) WORKLOAD TEST

In Figure 5, we evaluate the performance of SCARL with various intensity of workloads that are characterized by job arrival rates in Poisson distributions (average on X-axis). As shown, SCARL performs consistently better, showing 9.2 % improvement on average over SJF-P across all rates, while higher job arrival rates normally increase the waiting time of the jobs and accordingly the slowdown as well.

In addition, we evaluate the performance with various ratios of batch jobs in simulation workloads. In Figure 6, the X-axis denotes such a ratio that for a sequence of 200 jobs, 0.25 means that 50 jobs arrive at the initial timestep (time 0) while the other 150 jobs set to follow a given arrival rate in a Poisson distribution. Similar to the previous result, SCARL outperforms the others consistently. This result demonstrates the robustness of SCARL especially for the case of having many pending jobs since high ratios of batch jobs temporarily increase the number of pending jobs.

**TABLE 7. Dynamic cluster configurations.**

| Cluster Operation Configuration ( $\Delta \in \{1, 2\}$ ) |              |                                     |
|-----------------------------------------------------------|--------------|-------------------------------------|
| S-                                                        |              | $ M_{test}  - \Delta =  M_{train} $ |
| S+                                                        |              | $ M_{test}  + \Delta =  M_{train} $ |
| D-                                                        | During test, | $ M_{test}  - = \Delta$             |
| D+                                                        | During test, | $ M_{test}  + = \Delta$             |

**FIGURE 7. Dynamic condition configuration.**

### 3) DYNAMIC CHANGE TEST

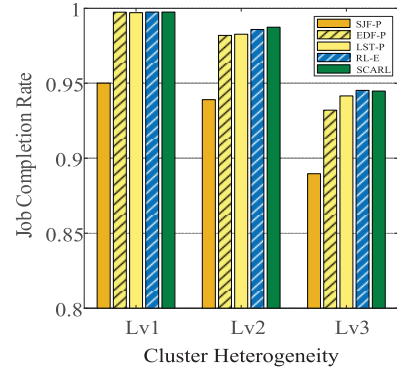
Here we evaluate the adaptability of the attentive embedding to time-varying conditions, which are described in Table 7. The static cases (denoted by S+ and S- in the table) indicate general cross-model tests where the cluster overall capacity, i.e., the number of machines  $|M|$ , sets differently for training and testing ( $|M_{train}|$  and  $|M_{test}|$  in the table). The dynamic cases (denoted by D+ and D-) indicate non-static cluster operating conditions where the cluster scale frequently changes during the operation of tests. For these tests only, we introduce a variant model, SCARL\_re, which is retrained according to each experiment scenario. Note that the SCARL model is commonly used for all regardless of the scenario.

Figure 7 depicts that SCARL performs with some resilience, showing the comparative performance with SJF-P and SCARL\_re which should be less sensitive for the dynamic conditions of each scenario.

### 4) DEADLINE TEST

We test another objective of RL-based scheduling, stipulating the cluster environment where each job is constrained in terms of its completion time, i.e., the deadline of real-time tasks. Figure 8 depicts the job completion rate. As shown, in most cases, SCARL and RL-E perform competitive with the well-known priority-driven algorithms for deadline, EDF-P and LST-P, and better than SJF-P.

This result is consistent with the resilience of RL-based schedulers for various scenarios and objectives. Interestingly, however, SCARL and RL-E rarely show a performance gap of significance especially on the Lv3 cluster, which is different from the previous result in Figure 4. In this test, the deadline constraint of jobs turned out to restrict the size and variance of outstanding jobs because the deadline-violated

**FIGURE 8. Deadline case.**

jobs can immediately be removed from the RL state. Therefore, we speculate that the global representation becomes less meaningful in this deadline test case than the slowdown tests, and consequently, SCARL and RL-E show almost the same performance.

### 5) VARIANTS OF FACTORED ACTION TEST

Here we compare SCARL with its variants, SCARL-O and SCARL-M. Recall that SCARL takes factored-action scheduling for job selection and machine allocation. SCARL-O conducts the job selection same as SCARL including the attentive embedding, but it exploits a heuristic method for the machine allocation, best fit packing. SCARL-M conducts the machine allocation same as SCARL including the attentive embedding, but it exploits a heuristic method for the job selection, SJF for the slowdown test and LST for the deadline test.

Figure 9 shows how SCARL achieves the performance gain by comparing with the variants over (a) SJF-P for the slowdown test and (b) EDF-P and LST-P for the deadline test. For comparison, we present the normalized values for slowdown and job completion rate with respect to the SCARL performance. For the slowdown test, SCARL outperforms the others including the two variants. It is observed that much gain came from the learning-based machine allocation policy in SCARL. For the deadline test, the performance gain of SCARL seems to be insignificant, as explained previously. Also note that gain of the learning-based machine allocation of SCARL is diminished.

It is important to appropriately represent the machines with different capabilities in the RL state when learning the machine allocation policy. Yet, the effect of global representations of machines and attentive embedding also depends on the diversity in cluster workloads. Indeed, the deadline constraint of jobs turns out to create a side-effect that reduces the variance of outstanding jobs.

### 6) POOL MODEL TEST

In addition to the Lv1-3 options previously tested, we also evaluate SCARL on the unified resource pool abstraction (Lv0). SCARL is intended for capturing the interdependent

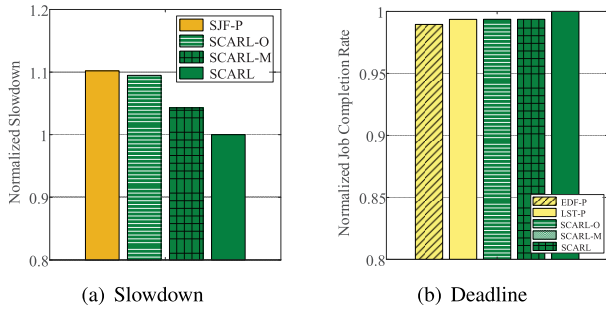


FIGURE 9. Variants of factored action. (a) Slowdown. (b) Deadline.

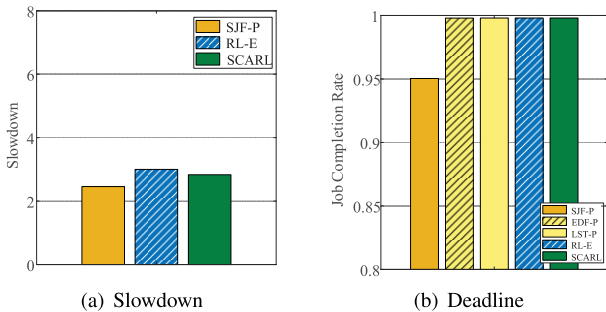


FIGURE 10. Unified resource pool. (a) Slowdown. (b) Deadline.

features of cluster entities and containing those on the global representation. With the resource pool abstraction, however, the global representation of SCARL might end up with less relevant information for scheduling, and this situation often generates overhead. The result is shown in Figure 10.

#### D. EXPERIMENT WITH OPERATION TRACE DATA

Here, we present the experiments with our test cluster for running jobs of synthetic workloads reproduced based on real cluster traces. We use the trace data available in [42] where each trace contains a sequence of operation logs collected from a Google data center and was used by several researchers, e.g., [21]. By definition, we treat tasks in the traces as jobs  $o_j \in O$  which might correspond to containerized application instances.

For working with the trace datasets, it is necessary to conduct several steps of data preprocessing to transform the original operation logs into the formatted feature data so that they can be used as the raw input to our simulation environment. (1) We estimate the machine features, i.e., resource capabilities  $rc_i$  by aggregating the resource requirements (CPU, memory, storage usages) of tasks on a same machine. (2) Based on the resource requirements of the tasks, we generate the job features (e.g., normalized resource requirements for CPU, memory, storage usages). In doing so, we fill the missing features (i.e., deadline  $dl_j$ , computation time  $ct_j$  that are not recorded in the trace) according to the parameters in Table 3. (3) We create a sequence of jobs using the event states in the trace; note that e.g., SCHEDULE(1) events indicate the job arrivals as described in [21] and [43] and so these events are used for generating non-preemptive

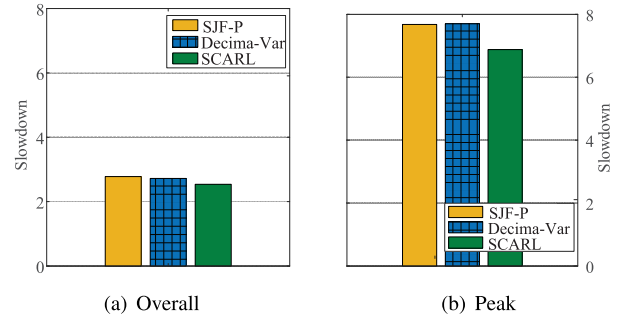


FIGURE 11. Test on a Kubernetes cluster with workload based on real trace. (a) Overall. (b) Peak.

jobs. We configure all of the job sequences consistent with the traces in different time scales in that the job arrivals during one unit of processing time in our simulation corresponds to those during 1,000,000 ms in the traces.

Through the data preprocessing steps above, we generate several datasets that are differently characterized as either overall or peak. Overall datasets simulate long-running average situations of cluster operation while peak datasets simulate highly dynamic situations of cluster operation where the job arrival rate is 10 times faster than the average rate. The duration of each overall and peak dataset is 500 and 50 timesteps respectively.

*Test With a Kubernetes Cluster:* For verifying our approach in a real cluster environment, we deploy SCARL in a Kubernetes cluster by exploiting the customizable structure of Kubernetes v1.15.21. Specifically, we implemented two additional modules to emulate a Kubernetes custom scheduler using Kubernetes-client APIs: (i) Scheduler frontend interface that translates RL actions of SCARL to Kubernetes commands for scheduling Pod and Node pairs (job and machine pairs in SCARL), and monitors scheduling-relevant events and status updates from Kubernetes controller, and (ii) Pod generator that creates a stream of Pods (a job sequence) from preprocessed trace datasets. We also test a state of the art RL-based cluster resource management. Since there have been rare RL-based studies that consider diversity and heterogeneity of both job and machines, we implemented a variant of Decima (namely, Decima-Var) based on [7]. The Decima-Var implementation becomes simplified compared to the original Decima in that it removes the task-level graph embedding function that handles multi-stage interdependent tasks within a job, which is not our concern; but it is extended with the Best Fit Packing heuristic for machine allocation. Note that the original role of machine allocation in Decima is to determine only the number of executors (machines).

Figure 11 illustrates the performance in slowdown on the workload that is generated based on real traces as explained above. In this experiment, we use a 5-node Kubernetes cluster deployed in our lab environment where the cluster sets to follow the same configuration of Lv3 heterogeneity in Table 4. We observe that SCARL reveals 8.6 % and 10.4 % improvement in slowdown over SJF-P for the overall and peak datasets respectively, while Decima-Var reveals 2.2 % and

0.2 % improvement over SJF-P. This result of testing on the Kubernetes cluster is consistent with the simulation results above. Recall that we consider a heterogeneous cluster as our target environment of RL-based schedulers, and accordingly, we develop an end-to-end RL structure that incorporates both job selection and machine allocation. As opposed to our approach, Decima focused on scalable embedding for complex data processing jobs, having no concern on cluster heterogeneity. Therefore, it should be noted that direct comparisons between our approach and Decima in terms of performance do not mean much. Through the result, we rather observe possible advantages of an end-to-end RL structure for dealing with both job selection and machine allocation.

## VI. CONCLUSION

In this paper, we proposed an RL-based scheduler focusing on a multi-resource heterogeneous cluster. Our approach SCARL efficiently learns to extract the relevant features of cluster operating conditions for combinatorial scheduling decisions through the iteration of attentive embedding tasks.

Our future direction is to adopt sequence learning models for presenting the factored actions more precisely. We also have been working on an RL-based scheduler for operating large-scale workloads over multiple clusters.

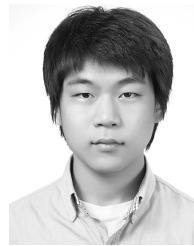
## REFERENCES

- [1] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with Borg," in *Proc. 10th Eur. Conf. Comput. Syst. (EuroSys)*, Bordeaux, France, Apr. 2015, Art. no. 18.
- [2] S. Chang, W. Han, J. Tang, G.-J. Qi, C. C. Aggarwal, and T. S. Huang, "Heterogeneous network embedding via deep architectures," in *Proc. 21st ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, Sydney, NSW, Australia, Aug. 2015, pp. 119–128.
- [3] Y. Hu, C. De Laat, and Z. Zhao, "Multi-objective container deployment on heterogeneous clusters," in *Proc. 19th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, Larnaca, Cyprus, May 2019, pp. 592–599.
- [4] A. Beloglazov and R. Buyya, "Energy efficient resource management in virtualized cloud data centers," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, Melbourne, VIC, Australia, May 2010, pp. 826–831.
- [5] X. Li, Z. Qian, S. Lu, and J. Wu, "Energy efficient virtual machine placement algorithm with balanced and improved resource utilization in a data center," *Math. Comput. Model.*, vol. 58, no. 5, pp. 1222–1235, Sep. 2013. doi: [10.1016/j.mcm.2013.02.003](https://doi.org/10.1016/j.mcm.2013.02.003).
- [6] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw. (HotNets)*, Atlanta, GA, USA, Nov. 2016, pp. 50–56.
- [7] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, Beijing, China, Aug. 2019, pp. 270–288.
- [8] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn. (ICML)*, Sydney, NSW, Australia, Aug. 2017, pp. 2430–2439.
- [9] Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, Paris, France, Apr./May 2019, pp. 505–513.
- [10] R. Li, Z. Zhao, Q. Sun, C. I. C. Yang, X. Chen, M. Zhao, and H. Zhang, "Deep reinforcement learning for resource management in network slicing," *IEEE Access*, vol. 6, pp. 74429–74441, 2018. doi: [10.1109/ACCESS.2018.2881964](https://doi.org/10.1109/ACCESS.2018.2881964).
- [11] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: Cognitive network management in sliced 5G networks with deep learning," in *Proc. IEEE Conf. Comput. Commun.*, Paris, France, Apr./May 2019, pp. 280–288.
- [12] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, Los Angeles, CA, USA, Aug. 2017, pp. 197–210.
- [13] C.-Z. Xu, J. Rao, and X. Bu, "Url: A unified reinforcement learning approach for autonomic cloud management," *J. Parallel Distrib. Comput.*, vol. 72, no. 2, pp. 95–105, Feb. 2012. doi: [10.1016/j.jpdc.2011.10.003](https://doi.org/10.1016/j.jpdc.2011.10.003).
- [14] N. Liu, Z. Li, J. Xu, Z. Xu, S. Lin, Q. Qiu, J. Tang, and Y. Wang, "A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Atlanta, GA, USA, Jun. 2017, pp. 372–382.
- [15] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-aware scheduling for heterogeneous datacenters," *ACM SIGPLAN Notices*, vol. 48, no. 4, pp. 77–88, Mar. 2013. doi: [10.1145/2499368.2451125](https://doi.org/10.1145/2499368.2451125).
- [16] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 127–144, Mar. 2014. doi: [10.1145/2541940.2541941](https://doi.org/10.1145/2541940.2541941).
- [17] Y. Xu, J. Yao, H. Jacobsen, and H. Guan, "Cost-efficient negotiation over multiple resources with reinforcement learning," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, Vilanova i la Geltru, Spain, Jun. 2017, pp. 1–6.
- [18] W. Wang, B. Li, and B. Liang, "Dominant resource fairness in cloud computing systems with heterogeneous servers," in *Proc. IEEE Conf. Comput. Commun.*, Toronto, ON, Canada, Apr./May 2014, pp. 583–591.
- [19] W. Liu, N. Chen, H. Li, Y. Tang, and B. Liang, "A heterogeneous cluster multi-resource fair scheduling algorithm based on machine learning," in *Proc. Int. Conf. Pioneering Comput. Sci., Eng. Educators*, Zhengzhou, China, 2018, pp. 548–559.
- [20] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, Chicago, IL, USA, Aug. 2014, pp. 455–466.
- [21] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput. (SoCC)*, San Jose, CA, USA, Oct. 2012, Art. no. 7.
- [22] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Comput. Surv.*, vol. 52, no. 5, Sep. 2019, Art. no. 97. doi: [10.1145/3326066](https://doi.org/10.1145/3326066).
- [23] A. Kliem and O. Kao, "The Internet of Things resource management challenge," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst.*, Sydney, NSW, Australia, Dec. 2015, pp. 483–490.
- [24] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache Hadoop YARN: Yet another resource negotiator," in *Proc. 4th Annu. Symp. Cloud Comput. (SOCC)*, Santa Clara, CA, USA, Oct. 2013, Art. no. 5.
- [25] *Kubernetes*. Accessed: Sep. 25, 2019. [Online]. Available: <https://kubernetes.io>
- [26] L. Thamsen, B. Rabier, F. Schmidt, T. Renner, and O. Kao, "Scheduling recurring distributed dataflow jobs based on resource utilization and interference," in *Proc. IEEE Int. Congr. Big Data (BigData Congr.)*, Honolulu, HI, USA, Jun. 2017, pp. 145–152.
- [27] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Boston, MA, USA, Mar. 2011, pp. 323–336.
- [28] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2014. *arXiv:1409.0473*. [Online]. Available: <https://arxiv.org/abs/1409.0473>
- [29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Long Beach, CA, USA, 2017, pp. 5998–6008.
- [30] C. Zhou, J. Bai, J. Song, X. Liu, Z. Zhao, X. Chen, and J. Gao, "Atrank: An attention-based user behavior modeling framework for recommendation," in *Proc. AAAI Conf. Artif. Intell.*, New Orleans, LA, USA, Apr. 2018, pp. 4564–4571.
- [31] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," 2018. *arXiv:1805.08318*. [Online]. Available: <https://arxiv.org/abs/1805.08318>
- [32] L.-C. Chen, Y. Yang, J. Wang, W. Xu, and A. L. Yuille, "Attention to scale: Scale-aware semantic image segmentation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 3640–3649.

- [33] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," in *Proc. 32nd Int. Conf. Int. Conf. Mach. Learn. (ICML)*, Lille, France, Jun. 2015, pp. 2048–2057.
- [34] K. Li, Z. Wu, K.-C. Peng, J. Ernst, and Y. Fu, "Tell me where to look: Guided attention inference network," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, Jun. 2018, pp. 9215–9223.
- [35] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia, "Relational deep reinforcement learning," 2018, *arXiv:1806.01830*. [Online]. Available: <https://arxiv.org/abs/1806.01830>
- [36] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," 2016, *arXiv:1611.09940*. [Online]. Available: <https://arxiv.org/abs/1611.09940>
- [37] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" 2018, *arXiv:1803.08475*. [Online]. Available: <https://arxiv.org/abs/1803.08475>
- [38] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," in *Proc. 28th Int. Conf. Neural Inf. Process. Syst. (NIPS)*, Montreal, QC, Canada, 2015, pp. 2692–2700.
- [39] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. Cambridge, MA, USA: MIT Press, 1998, vol. 2, no. 4.
- [40] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, 1992.
- [41] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," 2018, *arXiv:1807.02264*. [Online]. Available: <https://arxiv.org/abs/1807.02264>
- [42] J. Wilkes and C. Reiss. (2011). *Details ClusterData-2011-1 Trace*. [Online]. Available: <https://code.google.com/p/googleclusterdata/wiki/ClusterData20111>
- [43] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google cluster-usage traces: Format+ schema," Google Inc., Menlo Park, CA, USA, White Paper, Nov. 2014, pp. 1–14.



**MUKOE CHEONG** was born in Seoul, South Korea. He received the B.S. and M.S. degrees in software from Sungkyunkwan University, Suwon, South Korea, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree. His current research interests include vehicular network, cloud computing, and intelligent application.



**HYUNSUNG LEE** was born in Seoul, South Korea. He received the B.S. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2019, where he is currently pursuing the master's degree in electrical and computer engineering. His current research interests include cluster orchestration and reinforcement learning.



**IKJUN YEOM** received the B.S. degree in electronic engineering from Yonsei University, Seoul, South Korea, in February 1995, and the M.S. and Ph.D. degrees in computer engineering from Texas A&M University, in August 1998 and May 2001, respectively. He was with DACOM Company from 1995 to 1996, and also with Nortel Networks, in 2000. He was an Associate Professor with the Department of Computer Science, KAIST, from 2002 to 2008. He is currently a Full Professor with the Software Department, Sungkyunkwan University, Suwon, South Korea. His current research interests include AQM, congestion control, TCP, wireless networks, and future internet architecture.



**HONGUK WOO** was born in Seoul, South Korea. He received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer science from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he was with the Samsung Research of Samsung Electronics, as a Principal Engineer and a Vice President. Since 2018, he has been an Assistant Professor with the Department of Software, Sungkyunkwan University, Suwon, South Korea. His current research interests include intelligent application, analytic monitoring, cloud computing, and networked cyber-physical systems.

...