Received July 3, 2020, accepted July 14, 2020, date of publication July 16, 2020, date of current version July 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3009826

Virtual Cluster Deployment Model for Large-Scale Data Processing Jobs

YUNPENG CAO AND HAIFENG WANG
School of Information Science and Engineering, Linyi University, Linyi 276005, China
LinDa Institute of Shandong Provincial Key Laboratory of Network Based Intelligent Computing, Linyi University, Linyi 276000, China

Corresponding author: Haifeng Wang (gadfly7@126.com)

This work was supported in part by the Shandong Provincial Natural Science Foundation, China, under Grant ZR2017MF050, and in part by the Shandong Province Key Research and Development Program of China under Grant 2018GGX101005, Grant 2019GGX101003, and Grant 2019GNC106027.

ABSTRACT The virtual cluster needs to deploy virtual nodes with different computing modes for Large-scale data computing jobs including many tasks with different computing modes. There are resource unbalance among the virtual cluster subsets, which is consisted of virtual nodes with the same computing mode. So the resource utilization of virtual cluster is poor due to the time-varying and complexity of workloads. To solve these issues, a novel dynamic deployment model based on Docker was proposed to optimize resource configuration of virtual cluster. Hybrid deployment scheme includes coarse-grained deployment and fine-grained deployment in order to absorb the mutation of resource demand when processing the jobs encapsulated with complex computing logic. This deployment model can reduce job waiting time and improve performance by providing more resource for running jobs. The coarse-grained deployment mechanism has the ability of decreasing deploying overhead and improving stability from global optimal perspective. And the fine-grained deployment mechanism can increases the deploying accuracy from local optimal perspective. The experiments show that this model can improve the execution efficiency of job by 2.3 percent compared with static deploying scheme and improve the virtual CPU utilization about 7.9 to 25.9 percent compared with other approaches.

INDEX TERMS Large-scale data process, virtual cluster, dynamic deployment, Docker container.

I. INTRODUCTION

A. MOTIVATION

The large-scale data analytics jobs require a significant amount of computing power and memory capacity that can be only obtained via distributed computing. Virtual cluster deployed on cloud infrastructures have introduced many benefits compared to physical cluster, avoiding upfront investments and the ability to adapt execution environment to applications [1]. Virtual cluster is significant computing infrastructure for the large-scale data analytics jobs.

The Large-scale data analytics jobs process encapsulates a complex computing logic into independent or dependent stages that belong to different computing modes [2]. So the large-scale data analytics job covering different computing modes includes batch processing task, stream computing task, memory computing task, graphic computing task, etc. Batch computing is a typical mode from MapReduce including map phase and reduce phase. It is suitable to deal with tasks, which are offline data process task and sensitive to throughput. Memory computing needs to iteratively process the same big dataset, which has to be kept in memory to reduce I/O latency. Stream computing divides continuous data streams into bounded streams, which is very sensitive to latency. Graphic computing may be transmitted a sequence tasks with dependency relations based on directed acyclic graph. Each computing mode has its unique resource request characteristics. In short, the Large-scale data analytics jobs need virtual nodes with different computing modes. Therefore the virtual cluster configuration and virtual node deploying are important not only to improve performance of jobs but also to reduce resource waste.

Container is experiencing a rapid development with support from industry and being widely applied in Large-scale virtual resource management [3]. Recently due to the increasing popularity of Docker, there is a trend to run Hadoop, Spark and Flink applications on Docker containers to build virtual cluster [4]. However, little attention has been paid to

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

the resource deploying of virtual cluster based on Docker for large-scale data analytics jobs. However, the virtual cluster also exposed some drawbacks for large-scale data analytics applications. There are two issues need to be solved:

- The diversity of computing modes hindered the massive adoption of virtual cluster for large-scale data analytics applications. The virtual resource utilization rate is poor in computing process of Large-scale data analytics jobs. The researches find that the virtual resource utilization is below 17% due to the fact that virtual cluster managements for large-scale data process focus on MapReduce model rather than considering various computing modes.
- The virtual resource utilization is poor in the computing process of large-scale data analytics jobs. there exist various tasks with different computing modes. This requires that the virtual cluster needs to be divided into different subset to handle the tasks with different computing modes. These subsets of virtual cluster are responsible for different computing modes. However the workloads among different subsets of virtual cluster in real time are unbalance. This leads to the low utilization of virtual resource.

To overcome above the issues, the deployment model of virtual cluster based on Docker was proposed to scale in or out virtual nodes. This model can dynamically adjust the size of subsets in virtual cluster to balance the workload in real time and improve the performance of Large-scale jobs. It may be used by cloud service provider or cluster manager to optimize virtual resource configuration. It focuses on the resource configuration and management of virtual clusters to adapt to the workloads of large-scale data analytics process with different computing modes.

B. RELATED RESEARCH

Container is emerging as a promising virtual solution for users and has gained great popularity in industry. Docker with lightweight design and near-native performance turned containers into a mainstream technology.

Performance evaluation and analysis is the foundation for container application and optimization. There has been much research related to virtualization performance. Zhanibek *et al.* compared a range of existing container-based technologies for cloud computing system and evaluated virtualization overhead. The experimental results may provide some optimization direction of how to design Docker [4]. Truyen *et al.* evaluated the performance overhead of Docker Swarm and Kubernetes for deploying and managing NoSQL database clusters. They find a number of disadvantages of containers for NoSQL database workloads and propose some suggestions of network design to reduce virtualization overhead [5]. Andrew *et al.* investigated the use of containers in supercomputing and cloud system. They analyzed the performance overhead and highlighted a path for container technology in supercomputer system [6]. PyMon was designed to evaluate performance of Docker Swarm and Kubernetes in fog computing system [7]. Ahmed analyzed the performance of container in fog computing environments, which are often made of very small computers. They proposed three optimizations to reduce container deployment time [8]. Mavridis conducted experiments to investigate how the container performance is affected by the additional virtualization layer of virtual machine. Docker is applied to deploy and run containers on KVM hypervisor [9].

Container is increasingly applied into the large-scale data process field. The application of container in Large-scale data process is discussed as follows: Zeng *et al.* proposed an elastic big data processing system based on Spark and Docker to analyze large-scale data from networks perspective [10]. DCSpark is a framework that leverages Docker containers that allows users to run Spark applications. The results shows performance of DCSpark is very close to native Spark cluster [11]. Jin *et al.* evaluated performance of distributed data processing systems, Hadoop and Spark, on different container platforms. They revealed that the performance advantage of LXC and Docker are different with different workloads [12]. Ye *et al.* focused on performance tuning and modeling of Spark on Docker container driven cloud platform. They identified the key parameters affecting performance of Spark on Docker and detected the interference of performance between various big data applications [13]. Naik proposed Docker container-based technology for Hadoop and Pachyderm. He provided the detailed design scheme and performance comparison [14].

Container deployment optimization and virtual resource scheduling optimization are significant to virtualization technology. EC4Docker was designed and integrated with Docker Swarm to create auto-scaled virtual clusters of containers dedicated to tackle scientific workloads [2]. Octopus is a new cloud orchestrator that enables cloud independent infrastructure deployment at user level. It can help Docker Swarm deploy containers on hybrid clouds in order to balance workloads among different cloud systems [15]. Zhang *et al.* modeled container scheduling as an integer linear programming and proposed an effective and adaptive scheduler easily integrated into container orchestration frameworks [16]. Chang *et al.* aimed at developing a generic platform to facilitate dynamic resource-provisioning based on Kubernetes. This platform can dynamically monitor resource requirements and usage of running workloads. Then it adjusts the resource provision to prevent from resource overprovisioning and underprovisioning [17]. Lin *et al.* proposed a multi-objective container scheduling algorithm Multiopt that considers key factors of performance. They combined these factors into a composite function to solve the multi-objective optimization problem. And it was implemented based on Docker Swarm [18].

The current researches indicate it is a trend to apply container technology to build and manage large-scale data process infrastructure. However, on the one hand, the researchers rarely consider the difference of resource requirement among different computing modes. On the other hand, there lacks container deployment model for large-scale data analytics jobs to improve virtual resource utilization.

C. OUR CONTRIBUTION

The purpose of this work is to present a solution to dynamically adjust virtual nodes with low deployment overhead and thus obtain the efficient deploying performance and improve virtual resource utilization on two sides. The main contributions of this paper are as follows:

- Propose hybrid deployment model for virtual cluster with containers, which is in contrast to the static deployment strategies in applications. The benefits are many fold: to reduce unnecessary deploying complexity and to increase deployment efficiency. And to increase accuracy of virtual node deployment and resource utilization from the global perspective.
- Integer coarse-grained and fine-grained strategies to effectively reduce deployment overhead. What is crucial is that the hybrid deployment model has advantages of both coarse-grained strategy and fine-grained strategy. Additionally, introduce the prediction model to estimate resource demands before users submit jobs.
- Define the subcluster for the subset of virtual cluster and solve the resource unbalance among different subclusters in large-scale data analytics process. This can absorb the abrupt resource demand in specific subcluster and achieve the tradeoff of virtual resource among subclusters. On the other hand, it can optimize the performance through increasing virtual resource for specific jobs.

The rest of this paper is organized as follows. Section 2 gives the problem description and relevant terminology. Section 3 presents the detailed deployment approach including system design, model analysis and algorithms. Section 4 presents the experimental results and analysis. Section 5 concludes this paper.

II. PROBLEM DESCRIPTION

A. PROPOSED PROBLEM

In cloud data centers, the system will handle large number job requests of large-scale data analytics. However the Large-scale data analytics jobs comprise a set of dependent tasks, which belong to different computing modes. Here we called such job as complex job. For example, Fig. 1 provides a complex job, which has many tasks represented by different circles. This complex job has three computing stages. In the first stage, there are three stream computing tasks that need

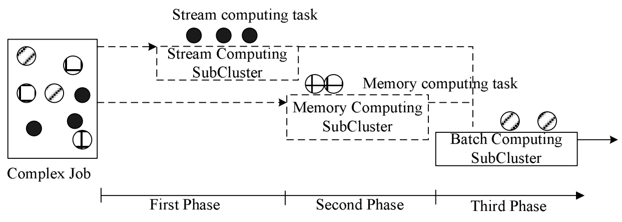


FIGURE 1. Complex job execution process.

to be submitted to virtual cluster with stream computing mode. Meanwhile two tasks need to be processed by memory computing virtual cluster. Note that the first stage and the second stage are concurrent execution relationship. Finally, the last two tasks complete in batch computing cluster.

Definition 1 (Complex Job): It can be defined as $J = (\mathrm{T}_\mathrm{s}^1, \mathrm{T}_\mathrm{s}^2, \mathrm{T}_\mathrm{m}^3, \mathrm{T}_\mathrm{m}^4, \mathrm{T}_\mathrm{b}^5, \ldots, \mathrm{T}_\mathrm{g}^\mathrm{k})$, T denotes the task in complex job, the symbols of subscript s, m, b, g represent stream computing task, memory computing task, batch computing task and graphic computing task, respectively. Such as, T_s is a task, which should be sent to virtual cluster running with stream computing mode. The superscript represents the task sequence. For example, T_s^1 must be executed before T_s^2.

The cloud system will assign and configure virtual resource when users submit complex job requests. As shown in Fig. 2, the upper part presents the requirement of complex jobs in different computing stages. The lower part of Figure 2 denotes the utilization of virtual resource. Black indicates virtual node is used and white is free. To handle different tasks with different computing modes, the virtual cluster is to be divided into different subsets. The virtual nodes subset is called subcluster, which is used to process tasks with specific computing mode. Such as batch computing subcluster, stream computing subcluster, memory computing subcluster and graph computing subcluster. Note that subcluster is the subset of virtual cluster in order to describe our work conveniently.

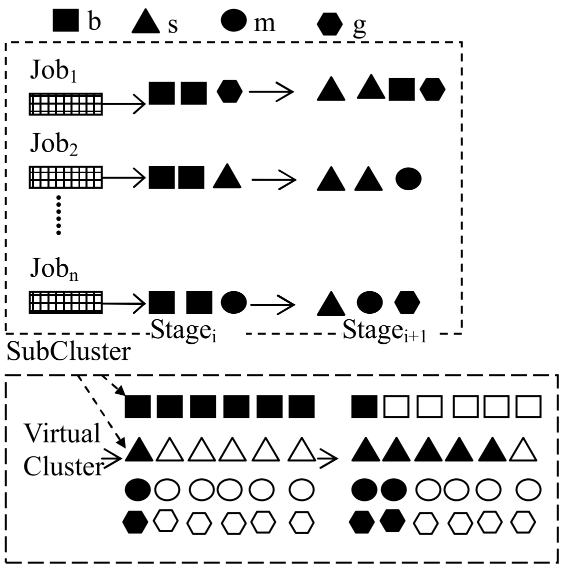


FIGURE 2. Virtual SubCluster for different computing modes.

Definition 2: **SubCluster** is a virtual node set, which is deployed to support the specific computing mode and enables appropriate tasks to execute on it. The virtual cluster consists of different subclusters with different computing modes. The virtual cluster denoted as VC is built based on physical cluster. Here the virtual cluster includes four subclusters, VC = {B, S, M, G}. B, S, M and G represent batch computing, stream computing, memory computing and graphic computing subcluster, respectively. B $= (b_1, b_2, \ldots, b_\mathrm{n})$, S $= (s_1, s_2, \ldots, s_m)$, M $= (m_1, m_2, \ldots, m_k)$, G $= (g_1, g_2, \ldots, g_l)$, where b_i, s_i, m_i, g_i denotes virtual node in subcluster.

Cloud service providers maintain virtual cluster to deal with the complex job request in real time. Resource scheduler generally deploys virtual nodes according to the highest load demands [19]. Therefore, many virtual nodes in subcluster remain relatively idle in most of time. This incurs the virtual resource utilization degradation. On the other hand, the request of jobs in real time is unbalance. Some subclusters may be idle, but others may have a heavy workload. As shown in Fig. 2, the batch computing nodes are occupied in stage i. The rest of subclusters are idle. When the next stage arrives, the stream computing subcluster is overload. The others are idle. So there is unbalance among these subclusters. There is no doubt that the optimal virtual cluster deployment strategy should have abilities to make the whole system have better load balancing effect and reduce resource waste. Therefore, it is necessary to design and implement an efficient and load-balancing virtual cluster deployment strategy for Large-scale data analytics jobs.

Our goal is to dynamically adjust virtual nodes of subclusters and to balance resource utilization. The central issue is virtual resource scheduling that is redeploy the configuration of subclusters according to the variation of workloads.

B. PROBLEM FORMULATION

To improve resource utilization of virtual cluster and overcome the limitation of existing deployment strategy, a dynamic virtual node scheduling is described as Equation 1.

$$f : Job(t) \rightarrow Vc(B, S, M, G)(t) \tag{1}$$

This is a function between complex jobs and virtual resource. The significant parameter is time t, which indicates virtual cluster configuration should be changed in response to variation of workloads. Figure 3 shows the scheduling process: in time i, there are n job requests accumulated in time window to be assigned into different subclusters.

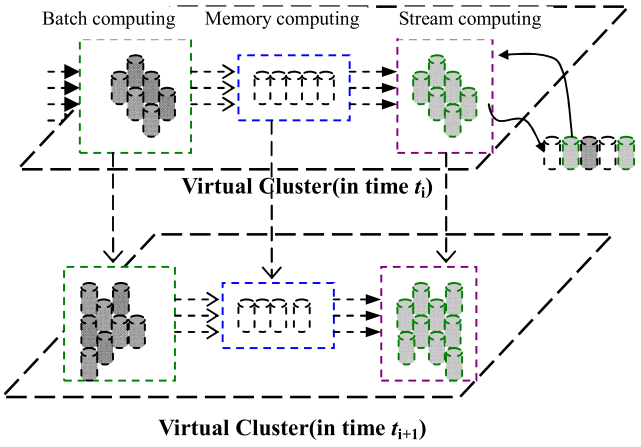

Batch computing
Memory computing
Stream computing
Virtual Cluster(in time t_i)
Virtual Cluster(in time t_{i+1})

FIGURE 3. Dynamic deployment model of virtual cluster.

The scheduling system adds or removes virtual nodes to build new subclusters in real time according to job requests. When the subcluster has a heavy workload, it will increase the capacity of this subcluster (scale-out). When the workload of subcluster is below a specific threshold, it may decrease the subcluster size (scale-in) [20]. As Shown in the lower part, in time $i + 1$, after the deployment, the subclusters complete the change of resource configuration. The better workloads balance among subclusters is achieved.

The virtual resource scheduling is to deploy virtual nodes including add, remove and update virtual nodes. Our deploying model uses a hybrid approach that employs prediction-driven and state-driven deploying strategies to achieve high accuracy and low overhead. So The Hybrid Dynamic Deployment Model (HDDM) of is defined as follows:

$$HDDM = <J, I, VM, E, R, f>, \tag{2}$$

where J denotes the job set of Large-scale data analytics. Each job can be described as a task sequence, such as $J_i = (T_b^1, T_s^2, T_b^3, T_m^4, \ldots,)$. The subscript symbols b, s, m, q denote batch computing, memory computing, stream computing, graphic computing, respectively. The sequence of tasks reflects the dependency relation in computing process. Such as $(T_b^1, T_s^2, T_b^3, T_m^4, \ldots,)$ means that the appropriate computing phases is as follows: T_b^1 is a batch computing task, that needs to deal with firstly; Secondly, T_s^2 is a stream computing task; Thirdly, T_b^3 is still a batch computing task, that needs to be processed in batch computing subcluster. Then T_m^4 is memory computing task and etc. I represents feature information of submitted jobs, such as the input data size and job type. VM is the virtual node set to deal with the submitted jobs. E is the triggering event set. R is the deployment rule and f is the function to map the triggering event E to deployment rule R.

The key problem is to maintain optimal VM according to resource requirements of complex job J. The main idea is that employs prediction-driven and state-driven deploying strategies. According to the feature information I, HDDM periodically obtains the rough virtual resource usage of submitted jobs by a prediction model. During the computing process, HDDM allows subcluster dynamically adjust virtual nodes based on the states of virtual resource. When the triggering event E is activated, the corresponding deploying rule R that is obtained by function f is executed to adjust the configuration of VM.

III. METHODOLOGY

Our dynamic deploying model is a hybrid deploying model, which is integrated with coarse-grained and fine-grained deployment mechanism.

A. MAIN IDEA

This section provides the detailed idea of hybrid deploying model that is to determine when and how to deploy virtual nodes for subclusters.

We provide two levels deploying mechanisms. The upper level is a coarse-grained deploying mechanism with time-triggering mode. The lower level is a fine-grained deployment mechanism with event-triggering mode. As shown in Fig. 4 the time sequence $(t_1, t_2, t_3, t_4, \ldots, t_n)$ is time signals for the upper deploying. When t_i comes, the ith upper deploying may be initiated. On the other hand, the lower deploying mechanism is applied in time interval $[t_i, t_{i+1}]$, which is called a scheduling window. The event signals are used to trigger deployment in scheduling window.

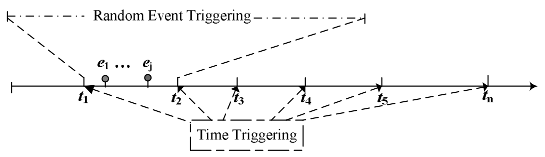


FIGURE 4. Deployment time.

The job sequence $(J_0, J_1, \ldots, J_m)$ in t_1 time is called job scheduling queue. The resource requirement of job is denoted as VM^{req} and the number of current available virtual nodes is denoted as VM^{cur}. Assume that the job scheduling queue has 20 complex jobs and VM^{cur} is 120 in time t_1. The coarse-grained deploying is prediction-driven method. It will obtain the resource requirements of this job scheduling queue VM^{req} by a resource usage prediction model. Assume that VM^{req} is 150. Then the coarse-grained resource scheduling will be initiated due to the fact that VM^{req} is greater than current available virtual resource. The dynamic deployment model will add 30 virtual nodes into virtual cluster to satisfy resource requirement. So the fine-grained deploying starts to work in the first job scheduling window $[t_1, t_2]$. The fine-grained deploying is state-driven method. The change of resource state can generate triggering events. As shown in Fig. 4 the event sequence $(e_1, e_2, \ldots e_j)$ can initiate the adjustment of virtual nodes in each subcluster in order to adapt to workload variation. When the first job scheduling window ends, the next coarse-grained deploying will be launched in time t_2. Assume this time VM^{req} is 130 and the virtual cluster has 145 available virtual nodes. So the dynamic deployment model will close 25 virtual nodes. In the second job scheduling window $[t_2, t_3]$, the fine-grained deploying continue to work by the event-triggering mode. This process goes on until all jobs complete.

B. DESIGN OF SYSTEM ARCHITECTURE

Fig. 5 describes architecture of the hybrid dynamic deployment system in cloud data center. It shows interaction between dynamic deployment model and other modules. The system includes job preprocess, resource usage prediction model, event generator, rules set and deployment controller. The deployment controller is key module, which is to manage and coordinate the coarse-grained and fine-grained deploying modules.

In the coarse-grained deploying process, the job preprocess module acquires information of resource requestment

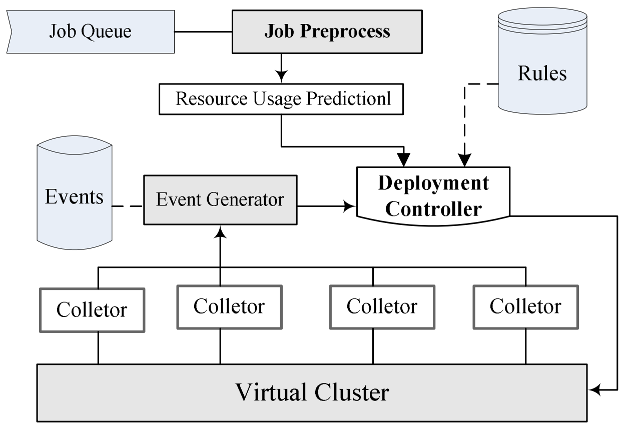


FIGURE 5. Dynamic deployment model of virtual cluster.

provided by users. It offers input data for the resource usage prediction model. Then this prediction model generates information to deployment controller for coarse-grained deployment. On the other hand, in the fine-grained deploying process, the collectors acquire state information of virtual nodes. By using the information monitored from collectors and event database, the event generator produces triggering events, which is transmitted to deployment controller to find the corresponding rules. Finally, the deployment controller will generate the dynamic deployment plan, which is transmitted to virtual cluster is to guide dynamically add, remove, migrate and update virtual nodes to update a new virtual cluster configuration.

C. COARSE- GRAINED DEPLOYMENT

1) RESOURCE USAGE PREDICTION

The coarse-grained deployment of HDDM is a prediction-driven virtual resource scaling method for multi-tenant cloud computing. The feature of submitted jobs is fed into an online resource usage prediction model to estimate the short-term resource demand profile.

Elman neural network is applied to build resource usage prediction model defined as Equation 3 [21].

$$E(X) = vm, \tag{3}$$

where X is the features of complex job provided by users. The output vm is number of virtual nodes for this job. The input vector $X = <x_1, x_2, x_3, x_4, x_5>$ of prediction model is the resource requirement information as shown in table 1.

To reduce complexity of training data and improve capacity of dealing with uncertainty, the requirements of resource are converted by fuzzy logic as shown in table 1 [22]. The first feature is job complexity. The greater of this value includes more types of computing tasks. For example, job j_1 has batch computing, stream computing and graph computing tasks. And job j_2 just includes stream computing and memory computing tasks. So the job complexity of j_1 is greater than j_2. The second feature is CPU intensive degree, which reflects the CPU occupancy of job. The third feature is memory

TABLE 1. Resource requirement features of jobs.

Feature	Value	Comment	Symbol
Job Complexity	High	High,Medium,Low	x_1
CPU Intensive	Medium	High,Medium,Low	x_2
Memory Intensive	Medium	High,Medium,Low	x_3
I/O Intensive	High	High,Medium,Low	x_4
Input_Data Size	200GB	Input Data Size	x_5

intensive degree, which reflects memory occupancy of job. The fourth feature is I/O intensive degree, which reflects I/O occupancy of job. When users submit jobs, they just need to distinguish the CPU-intensive job, memory-intensive job or I/O-intensive job. The users provide rough values for each feature. The last feature is input data size x_5. This significant feature reflects workload size of submitted jobs.

The Elman neural network is a partial recurrent network based on basic structure of BP network [23]. The input layer, hidden layer and output layer are the feed-forward loop. However Elman neural network has back-forward loop mechanism, which employs context layer as shown in Fig. 6. Note that the prediction model is to obtain the approximation of virtual resource demand rather than to pursue accuracy. There are two reasons to select Elman neural network for resource usage prediction: ① the virtual resource usage is typical time series data. Then the resource demand of job request is a typical time-varying data. The partial recurrent structure of Elman network and dynamic characteristics provided only by internal connection make Elman neural network prior to static feed-forward network or other artificial neural networks [24]. It is better to deal with time-varying data and enhance the global stability of network. ② the context layer is sensitive to the history of input data while the connections between context layer and hidden layer. This enhances learning capacity for dynamic time-varying data.

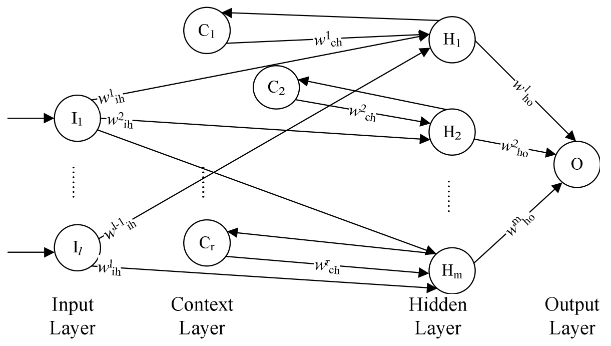


FIGURE 6. Elman neural network structure.

The topology of Elman neural network is designed as follows: the number of neurons in output layer r is the amount of virtual nodes, so the output layer node is $1(r = 1)$. The number of input layer n is the number of features extracted from jobs, so the input layer nodes is $5(n = 1)$. The number of hidden layer node m is obtained by the Equation 4.

$$Min\,(n, r) < m \leq 2n + 1, \tag{4}$$

Here $1 < m \leq 11$ and we select $m = 10$. Then the number of context layer node is set 10. The output of hidden layer is calculated by equation 5.

$$h(t) = \xi(\sum w^{in}_{ch} c(t) + \sum w^l_{ih}(u(t-1))), \tag{5}$$

where $h(t)$ denotes the output of hidden layer, $c(t)$ represents the output of context layer and $c(t) = h(t-1)$. $u(t-1)$ is the input in $t-1$ time. w_{ch} is the weight between context layer and hidden layer. w_{ih} is the weight between input layer and hidden layer. $\xi(x)$ is activation function defined as equation 6 [25].

$$\xi(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

The output of Elman neural networks is obtained by equation 7.

$$O(t) = \zeta(\sum_{n=1}^{N} w^n_{ho} h(t)), \tag{7}$$

$$\zeta(x) = kx + b, \tag{8}$$

$\zeta(x)$ is a linear function that is the weight between hidden layer and output layer. The output of Elman neural networks is floating number in [0,1]. Note that this value needs to be multiplied by the maximum of virtual node. Such as, the prediction value of resource requirements is 0.34 and the virtual node maximum is set 100. Then the final resource requirement is 34 virtual nodes.

Training data set is collected as follows: we selected many different Large-scale data process jobs with various data size. We choose the tradeoff between performance and virtual resource utilization as training objective. The desired number of virtual nodes assigned to each job is obtained according to the optimization objective by experiments. So one training data is obtained, for example, $E(<x_1, x_2, x_3, x_4, x_5> = <0.3, 0.6, 0.6, 0.3, 0.27>) = 8$. Then we have a training data set after many experiments.

2) DEPLOYING STRATEGY

The scheduling time of jobs is to determine when to initiate the coarse-grained deployment. When the last scheduling cycle ends, the time signal will trigger the next coarse-grained deployment. So the scheduling cycle is significant parameter that is determined by expert's experience and statistics method. When the scheduling signal appears, the deployment model will compare the resource requirement and current available virtual nodes. When the resource usage value exceeds specific thresholds, the virtual cluster will launch coarse-grained deployment of virtual nodes.

The following issue is to deploy the virtual nodes to different subclusters. Firstly, we introduce the resource utilization ratio $\mathrm{R_u}$.

$$R_u = \lambda_1 U_{CPU} + \lambda_2 U_{Mem} + \lambda_3 U_{IO} \tag{9}$$

U_{cpu}, U_{Mem} and U_{IO} are the average utilization of CPU, Memory and I/O. Each subcluster has different weight. To memory computing and graph computing modes, these kinds of jobs consume massive memory resource. So the weight of memory should be set higher value. However, the subcluster of batch computing needs to be given higher weight for I/O resource utilization due to high data throughput. The subcluster of stream computing ought to be set high weights for CPU and I/O resource utilization. The weights of subclusters are as follows: {b(0.3,0.2,0.5), s(0.4,0.2,0.2), m(0.2,0.5,0.3), q(0.1,0.5,0.4)}. For example, the average CPU, memory and I/O utilization of subcluster running for stream computing tasks are 0.45,0.23,0.68. Then the final resource utilization is calculated as follows.

$$R_u = 0.4 \times 0.45 + 0.2 \times 0.23 + 0.2 \times 0.68 = 0.362 \tag{10}$$

The second step is to determine how to deploy for different subclusters according to resource utilization. Assume that the number of adjustment virtual nodes is ΔVM. And ΔVM_i is the number of adjustment virtual nodes for different subclusters.

$$\Delta VM = \left|\mathrm{VM}^{\mathrm{req}} - \mathrm{VM}^{cur}\right| \tag{11}$$

$$\Delta VM_i = \frac{R_u^i}{\sum_i R_u^i} \quad (i = b, s, m, q),\ \Delta VM > 0 \tag{12}$$

$$\Delta VM_i = \frac{1 - R_u^i}{\sum_i R_u^i} \quad (i = b, s, m, q),\ \Delta VM < 0 \tag{13}$$

When the resource requirement is larger than current available resources, the new virtual nodes may be deployed to the different subclusters based on Equation 12. On the other hand, the virtual nodes may be removed from subclusters as shown in Equation 13 if the resource requirement is less than the number of available virtual nodes.

We should solve an important issue in the deployment process. The resource requirement of jobs is roughly equivalent to the available resource of whole virtual cluster. However, there exists unbalance among different subclusters. Such as $\mathrm{VM^{req}} = 38$ and $\mathrm{VM^{cur}} = 41$, the resource requirements and available resource of subclusters is as following: $\mathrm{vm^{req}} = (15, 8, 6, 9)$, $\mathrm{vm^{cur}} = (8, 16, 7, 10)$. So this strategy is to adjust the number of different subclusters to obtain resource balance. Note that the resource request sequence is $\mathrm{vm_i^{req}} = (15, 8, 6, 9)$, $\mathrm{vm_{i+1}^{req}} = (8, 15, 6, 10)$ and $\mathrm{vm_{i+2}^{req}} = (16, 9, 6, 10)$. Then the subcluster $\mathrm{c_1}$ and $\mathrm{c_2}$ may frequently exchange resource status. We called this phenomenon as deployment oscillation, which incurs extra deployment cost. To solve deployment oscillation, lazy factor was introduced to this strategy in order to delay deployment time.

D. FINE-GRAINED DEPLOYMENT

The fine-grained deployment mechanism is designed to adjust virtual node configuration in job scheduling window based on triggering events.

Definition 3: Triggering events are consisting of signals and appropriate thresholds. The signals include resource monitoring signals and resource requirement signals. Resource monitoring signals are generated by monitoring the utilization of CPU, memory, disk and networks. Resource requirement prediction model generates resource requirement signals when the resource requirement exceeds specific thresholds. These signals are able to capture resource status of jobs in computing process. The triggering event is to trigger redeployment of virtual cluster only when signals are above or below specific thresholds.

In the job scheduling window, triggering event is to determine when to launch deployment. The deployment rules are responsible for how to deploy virtual nodes. So the fine-grained deployment can be defined as a function between triggering set and deploying rule set.

$$f : \{E \to R\}, \tag{14}$$

where $E = \{e_1, e_2, \ldots, e_{\mathrm{i}}\}$ is triggering event set including performance events and execution environment events. $R = \{r_1, r_2, \ldots, r_{\mathrm{m}}\}$ is deployment rule set generated by experts. Then the virtual subclusters deployment is described as follows: the virtual cluster in i time is $\mathrm{VC_i} = \{\mathrm{C_i^{mr}}, \mathrm{C_i^{m}}, \mathrm{C_i^{s}}, \mathrm{C_i^{q}}\}$, which includes different subclusters. After the deployment, the virtual cluster in $i + 1$ time is changed into $\mathrm{VC_{i+1}}$.

$$f : VC_i \frac{R_i(r_1, r_2, \ldots, r_m)}{E_i(e_1, e_2, \ldots, r_k)} \to VC_{i+1} \tag{15}$$

So the event-triggered model is kernel of the fine-grained deploying mechanism and can be defined as follows.

$$\begin{cases} S^k = \{s_0, s_1, \ldots, s_n\} \\ e^k = f(S^k, R, \varepsilon) \end{cases} \tag{16}$$

$\mathrm{S^k}$ is signal sequence in time k. $f(S^k, R, \varepsilon)$ is the event-triggering function. $R = \{r_1, r_2, \ldots, r_{\mathrm{i}}\}$ is deployment rule set and $\varepsilon = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_{\mathrm{j}}\}$ is parameter set of event rule. The parameter set is listed in Table 2. The triggered rule in R is described as follows:

r_1: if $z_0 < \delta_1$ then v_1; r_2: if $z_1 > \delta_2$ then v_2.

Finally, the event is activated by Equation 17.

$$e^k = \begin{cases} 1, & f(S^k, R, \varepsilon) > \delta \\ 0, & otherelse \end{cases} \tag{17}$$

$\delta = (\delta_1, \delta_2, \delta_3, \delta_4)$ is triggering threshold set, which affects sensitive of deploying model. As shown in Equation 17 and Table 2, the event will be produced when some signals are above or below certain thresholds. To improve robustness of model, the event triggering thresholds are in the form of interval. For example, δ_1 and δ_2. When the resource requirement signals are not in the interval $[\delta_1, \delta_2]$, the triggering events

TABLE 2. Triggering event.

Symbol	Event	Comment
e_1	Resource_req > δ_1	Add nodes When requirement exceeds upper limit;
e_2	Resource_req < δ_2	Delete nodes When requirement is below lower limit;
e_3	$\sum_{i=1}^{4} CPU^i = \sum_{i=1}^{4} CPU\_Req^i$	Update Configuration When the number of virtual CPU in the four subclusters is equal to the number of requirement virtual CPU;
e_4	Resource utilization rate < δ_3	Suspend nodes When resource utilization rate is below lower limit;
e_5	Resource utilization rate > δ_4	Wake up nodes When resource utilization rate exceeds upper limit.

will be generate and virtual cluster may adjust resource configuration. So the size of interval is significant for the deploying model. When the interval is too small, virtual cluster may frequently adjust virtual nodes. This will improve deploying overhead. When the interval is too large, the deploying model is insensitive to the variation of workloads. This may reduce accuracy of deploying model. Here we obtained the optimal triggering thresholds by experiments, whose goal is to achieve tradeoff between deploying overhead and accuracy.

The deployment rules are divided into three groups: the status management to adjust virtual node status, node management to decide the configuration and size of virtual cluster and subcluster management used to determine the status or configuration of subclusters.

TABLE 3. Deployment rules.

Type	Rule	Case
Status management	Start Node	Start(VM_i,...VM_n)
	Close Node	Close(VM_i,...VM_n)
	Suspend Node	Suspend(VM_i,...VM_n)
	Restore Node	Restore(VM_i,...VM_n)
	Migrate Node	Migrate(VM_i,...VM_n)
Configuration management	Add Node	Add(VM_i,...VM_n)
	Delete Node	Del(VM_i,...VM_n)
	Update Node	Update(VM_i,...VM_n)
Subcluster management	Add Subcluster	AddGroup(VM_i,...VM_n, VC)
	Remove subcluster	RemoveGroup(VM_i,...VM_n, VC)

E. DEPLOYMENT ALGORITHM

In this section, we describe the detailed process of dynamic deployment for virtual cluster. The pseudocode of deployment algorithm has been given as follows.

Algorithm 1 Dynamic Deployment Algorithm

```
1. Initialize job scheduling queue $J$, $t$ and Lazy_factor;
2. For each $t = t_i$ (i = 1,2,...,n) Do
3.   If ($J$ = full) Do
4.     If $\sum vm^{req} \neq \sum vm^{cur}$ Then
5.       Adjust_Globalcluster();
6.     EndIf
7.     If $\sum vm^{req} \approx \sum vm^{cur}$ Then
8.       Adjust_Subclusters();
9.     EndIf
10.  EndIf
11.  Scheduling all jobs in $J$ and clear job queue;
12.  While ($t_i < t$ and $t < t_{i+1}$) Do
13.     Execution computing
14.     Call Fine-grained Deployment( )
15.  EndWhile
16.  Repeat Receive new job and AddQueue $J$
17.  i = i + 1
18. EndFor
```

In algorithm 1, the outer loop is the coarse-grained deploying. It is time for deploying, namely the current time $t = t_i$ (Line2). The coarse-grained deployment launches. The first step is to roughly predict current resource requirement by resource prediction model. When the resource requirement is far greater than current available resource, (Line3), we use Adjust_Globalcluster() to redeploy the configuration of virtual nodes (Line5). When the number of requirement virtual nodes equals the available virtual nodes, but the gap among different subclusters is large (Line7). We use Adjust_Subclusters to deal with the unbalance of resource in subclusters (Line8). After the coarse-grained deploying, the jobs in queue are scheduled (Line11). The virtual cluster starts to enter the computing process. The fine-grained deployment algorithm begins to work in the interval $[t_i, t_{i+1}]$ (Line12-15).

Adjust_Globalcluster is described as follows.

In algorithm 2, virtual cluster adjusts the virtual nodes of subclusters according to the proportion of resource utilization. When the virtual cluster needs to scale-out, this algorithm increases the number of virtual nodes for each subcluster by Equation 12 (Line1-5). When the virtual cluster needs to scale in, each subcluster removes the virtual nodes by Equation 13 (Line 6-10).

In Adjust_Subclusters algorithm, lazy factor is an important parameter, which can effectively eliminate deployment oscillation. This is because that the unbalance of virtual resource among different subclusters appears, this algorithm will delay resource optimization rather than immediately launch deployment. When the unbalance lasts for a period

Algorithm 2 Adjust_Globalcluster Algorithm

```
1. If $vm^{req} > vm^{cur}$ Then
2.    For each subcluster $c_i$ Do
3.       $c_i += (vm^{req} - vm^{cur}) \times R_u^i \Big/ \sum_i R_u^i$;
4.    EndFor
5. EnfIf
6. If $vm^{req} < vm^{cur}$ Then
7.    For each subcluster $c_i$ Do
8.       $c_i -= (vm^{cur} - vm^{req}) \times (1 - R_u^i) \Big/ \sum_i R_u^i$;
9.    EndFor
10. EndIf
```

Algorithm 3 Adjust_Subclusters Algorithm

```
1.    If lazy_factor < threshold $\eta$ Then
2.       For each $c_i$ in subclusters Do
3.          If $vm^{cur}(c_i) < vm^{req}(c_i)$ Then
4.             AddNode($vm^{req}(c_i) - vm^{cur}(c_i)$)
5.          EndIf
6.       lazy_factor = lazy_factor + 1
7.    EndIf
8.    Else
9.       Call BalanceVC( )
10.   EndElse
```

time, BalanceVC is called to optimize the resource unbalance of subclusters.

Algorithm 4 BalanceVC Algorithm

```
1.  Initialize Balance Queue BQ
2.  For each $c_i$ in subclusters Do
3.     If $vm^{cur}(c_i) > vm^{req}(c_i)$ Then
4.        EnBQ($vm^{cur}(c_i) - vm^{req}(c_i)$)
5.     EndIf
6.  EndFor
7.  While Queue BQ is not Empty Do
8.     For each $c_i$ in subclusters Do
9.        If $vm^{req}(c_i) > vm^{cur}(c_i)$ Then
10.        Do
11.          $v_i$ = DeBQ()
12.          Addsubcluster($c_i$, $v_i$)
13.        Until($vm^{req}(c_i) = vm^{cur}(c_i)$ or BQ = empty)
14.     EndFor
15.  EndWhile
```

BalanceVC is to obtain the tradeoff of resource among subclusters. The first step is to find the redundant virtual nodes in subclusters (Line 2-6). Then the redundant virtual nodes are added to the subclusters, which lacks virtual resource (Line 8-14).

Algorithm 5 Fine-Grained Deployment Algorithm

```
1. Initialize Virtual subclusters {$VC_i^{mr}$, $VC_i^{m}$, $VC_i^{s}$, $VC_i^{q}$};
2. Initialize Event queue E;
3. For each signal period Do
4.    Collect signals from {$VC_i^{mr}$, $VC_i^{m}$, $VC_i^{s}$, $VC_i^{q}$};
5.    Generate event $<e_i, e_{i+1}, \ldots, e_{i+k}>$, and E+ = $(e_i, e_{i+1}, \ldots, e_{i+k})$
6.    While E is not Empty Do
7.       $e_i$ = deQuery(E);
8.       $r_i$ = Query(R, $e_i$)
9.       Execute deploying by $r_i$
10.   EndWhile
11. Endfor
```

In Fine-grained Deployment algorithm, the first step is to generate events based on the collected signals (Line 3-5). The second step is to extract the events from event queue and query the deploying rules by events (Line 6-8). The last step is to execute the accordingly deploying events.

The virtual nodes with different computing modes are generated and saved as image files of docker on the harddisk. In practice, deploying a virtual node is to load a image file from the local storage device. So the harddisk of physical machine stores many image files in advance. This method can save the deploying time compared with load from remote image sever.

F. MODEL ANALYSIS

This section discuss deployment overhead of proposed model. Figure 7 provides static deployment scheme, which only needs to be deployed once. So the static deployment model is consisted of T_s and T_c. T_s is the time of virtual cluster deployment and T_c is the computing time of jobs. Compared with static deployment scheme, the advantage of proposed scheme is that it has more opportunities to adjust resource configuration of virtual cluster in order to improve resource utilization. However this scheme increases deployment overhead, which may degrade performance of virtual cluster. As shown in Fig. 7, this computing process is divided into many cycles. Each cycle is consisted of T_{dc}, T_{df} and T_c. T_{dc} is the time of coarse-grained deployment before the

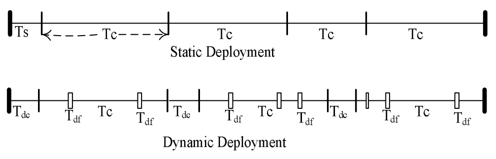


FIGURE 7. Comparison of two deployment schemes.

computing of jobs. T_{df} is the time of fine-grained deployment, which occurs in the computing of jobs.

Assume that a job queue running with two different deployment schemes, the total computing time is calculated by the equation 18.

$$\begin{cases} T_{total}^{s} = T_s + T_c^s \\ T_{total}^{d} = T_{dc} + T_c^d + k \times T_{df} \end{cases} \quad (k \geq 0, T_c^s \neq T_c^d), \quad (18)$$

where k is the number of triggering events. Note that the computing time of jobs with different deployment schemes are different. This is because that the dynamic deployment scheme may reduce job waiting time and improve performance by resource optimization.

In practice, the deploying overhead is mainly from the load of Docker image file in the coarse-grained deploying phase. Here the cycle length in coarse-grained deployment is set to 10 minutes. And the setting of this parameter is from expert experience. The deploying overhead has been measured in each coarse-grained deployment period. Each deploying with 192 virtual nodes takes 1.92s. Note that the dynamic deploying system has good expansibility. The deploying time will not increase with the virtual cluster size increment due to the fact that the dynamic deploying model is running in parallel mode.

The finer-grained deployment mechanism may improve accuracy of virtual resource utilization. However it may incur extra performance loss and oscillation of resource status, which is to frequently exchange resource status. On the other hand, the coarser-grained deployment mechanism with poor accuracy of deployment may reduce the oscillation of resource status and control the performance loss within acceptable range. So our hybrid dynamic deployment model achieves the advantages of coarse-grained and fine-grained deployment mechanism. It not only improves deployment accuracy but also reduce deployment overhead.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

A. EXPERIMENT SETUP

We built a virtual cluster in our university physical cluster, which includes 70 physical machines. Each physical machine with Intel i7 8700 6 cores CPU and 32G memory ran Ubuntu Xenial 16.04. Docker1.7.1 was used for server virtualization. To simplify resource scheduling, each virtual node has the same resource configuration, namely each node has equal CPU and memory. Each physical machine can create three Docker containers, which is assigned two CPU cores by CPU-Share proportion parameter in docker [26]. So the maximum size of virtual cluster is 210 virtual nodes. Docker Swarm was used to manage docker containers by executing management scripts. CAdvisor monitors the performance of virtual cluster and save the monitored data to influxDB, which is sequential database and support the interface of CAdvisor [27]. On the other hand, Hadoop running on virtual nodes deals with jobs of batch computing. Spark is responsible for memory computing jobs and Spark Streaming is to complete stream computing jobs. Spark GraphX is to handle graphic computing jobs [28]. We prepared these different mirror files of Docker, which include Hadoop system, Spark system. Each physical machine keeps various type mirror files in order to create different types of virtual nodes. The artificial workloads are used in experiments due to the fact that we can accurately verify the model.

We selected three benchmark models to compare our hybrid dynamic deployment model (HDDM). The first benchmark model originates from CodeCloud system, which provides a high-level declarative language to express the requirements of applications [31]. The application developers give the requirement information to Cloud providers. Then Cloud provider aggregates the requirement data from many users and allocates virtual nodes for jobs. So this deploying method is static deployment model denoted as SDM. The second benchmark model is the coarse-grained dynamic deployment model, which is a prediction-driven elastic resource scaling system for multi-tenant cloud computing. The typical prediction-driven virtual resource scaling model is CloudScale [32]. It is difficult to repeat the experiments of CloudScale. So we used our coarse-grained dynamic deployment scheme as the representative of prediction-driven deploying method. The third benchmark method is the fine-grained deployment scheme, which is a typical state-driven deploying model. The coarse-grained dynamic deployment and fine-grained dynamic deployment schemes are denoted as CDD and FDD, respectively.

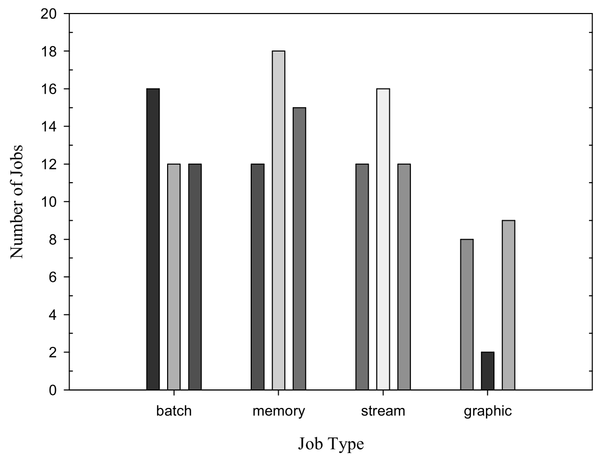
B. EFFECTIVENESS OF HDDM

This subsection aims to validate the effectiveness of hybrid dynamic deployment model (HDDM). The first work is to demonstrate that HDDM has the ability of dynamically adjusting configuration of virtual nodes to adapt to time-varying workloads. The second work is to show the advantage of HDDM. We made use of artificial workloads in experiments. The reason is that artificial workloads deliberately constructed can trigger the coarse-grained deployment mechanisms quickly. However, the real-world workloads may take a long time to see the results.

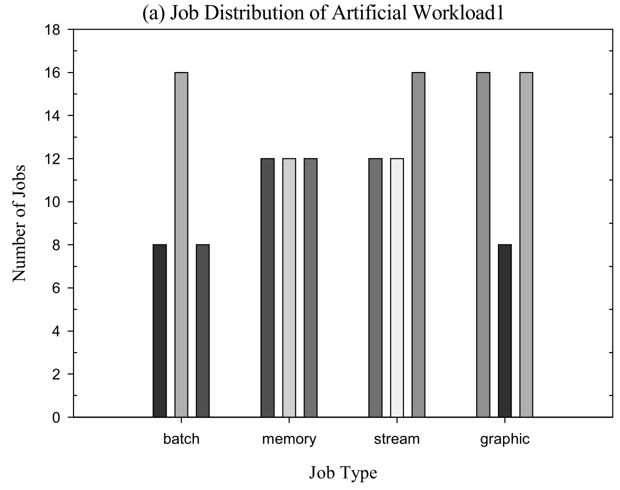
The approach of generating workloads is as follows: we select four types of jobs for artificial workloads. The job types are batch computing, memory computing, stream computing and graphic computing. Each type of job needs to be processed by the corresponding subcluster. We generate two artificial workloads. Each workload includes three job scheduling windows. One job scheduling window has 48 jobs. The computing process for the 48 jobs is called a stage. Then each artificial workload has three computing stages. Table 4 provides the detailed information of artificial workloads. As shown in Table 4, each stage lists the proportion of job types. Such as, the first stage of workload1 is 16:12:12:8, namely this stage has 16 batch computing jobs, 12 memory computing jobs, 12 stream computing jobs and 8 graphic computing jobs. Figure 8 provides the job distribution of artificial workloads. Such as, the first column of

TABLE 4. Artificial workloads.

ID	First Stage	Second Stage	Third Stage
Workload1	16:12:12:8	12:18:16:2	12:15:12:9
Workload2	8:12:12:16	16:12:12:8	8:12:12:16



(a) Job Distribution of Artificial Workload1



(b) Job Distribution of Artificial Workload2

FIGURE 8. Comparison with different artificial workloads.

Fig. 8(a) shows the variation of batch computing jobs in the three stages. The optimal virtual resource allocation of each job is about 2 to 5 virtual nodes by experiments.

The initial configuration of virtual cluster is set as follows: The initial configuration of virtual nodes is four times of the computing jobs. Such as, the jobs in the first stage of workloads1 are 16:12:12:8. Then the virtual cluster has 64 batch computing nodes, 48 memory computing nodes, 48 stream computing nodes and 32 graphic computing nodes.

Fig. 9 shows the comparison of HDDM with the benchmark models. x axis represents the computing process. And y axis indicates the number of virtual node adjustments. Fig. 9(a) demonstrates the number of adjusting virtual nodes in HDDM with workload1. In the begin stage, HDDM doesn't launch coarse-grained deployment due to the fact that the first job sequence in workload1 is 16:12:12:8. The resource requirement of this job sequence matches with the initial configuration of HDDM. Note that there exist 5 times deployments in the first stage. These noise deployments are triggered by fine-grained event deployment scheme in HDDM. However, HDDM launch two times adjusting activities in the 150th minute and the 310th minute, which are the beginning of computing for the second and third computing stages. The two deploying behaviors adjust more virtual nodes. This shows that the coarse-grained deployment scheme in HDDM can effectively optimize resource configuration. On the other hand, the fine-grained deployment scheme in HDDM just adjusts a small number of virtual nodes in the computing process. This result shown in Fig. 9(a) illustrates that HDDM can dynamically adjust resource configuration according to workloads variation. At the beginning of computing stage, the coarse-grained deployment scheme starts to work and complete the mainly allocation of virtual resource. So this causes more virtual node adjustments. On the contrary, the fine-grained deployment scheme in HDDM is responsible for the fine-tuning of virtual nodes in computing process.

Fig. 9(b),(c) shows the comparison of HDDM and CDD with workload2 to verify the advantage of HDDM, which is the ability of solving deployment oscillation. The resource requirement of first job sequence in workload2 is 8:12:12:16. In the first stage, there are 8 batch computing jobs and 16 graphic computing jobs. However, the initial configuration of virtual cluster is 64:48:48:32. The current available resources have 64 batch computing nodes and 32 graphic computing nodes. So it is obviously unbalance between batch computing subcluster and graphic computing subcluster. Note that the number of batch computing jobs in workload2 is changed from 8 to 16 in the second stage and is changed from 16 to 8 in the third stage. And the number of graphic computing jobs has contrary trends. Fig. 9(b) demonstrates that HDDM doesn't launch coarse-grained deploying due to lazy deployment mechanism to avoid deployment oscillation in the first two stages. It just depends on the fine-grained deployment scheme. When the third stage arrives in the 300th minute, HDDM begins to adjust virtual nodes by coarse-grained deployment scheme. But Fig. 9(c) shows that CDD initiates three virtual nodes adjustments in beginning of three computing stages. This results illustrate it is difficult for CDD without lazy mechanism to deal with deploying oscillation of virtual node.

Fig. 9(d) shows the comparison of HDDM and FDD with workload2. To make the results clearer, we monitored the adjustments of virtual nodes in graphic computing subcluster. To FDD scheme, the number of virtual nodes rises to 16 in the 50th minute. Then it drops to 9 in the 150th minute. Finally it reaches 16 again in the 300th minute. Due to the gap between the resource requirement and available resource, FDD triggers a number of virtual node adjustments at the beginning of each computing stage. Compared with HDDM, it appears two distinct resource scaling oscillations, which causes more performance loss. Therefore HDDM can effectively avoid the

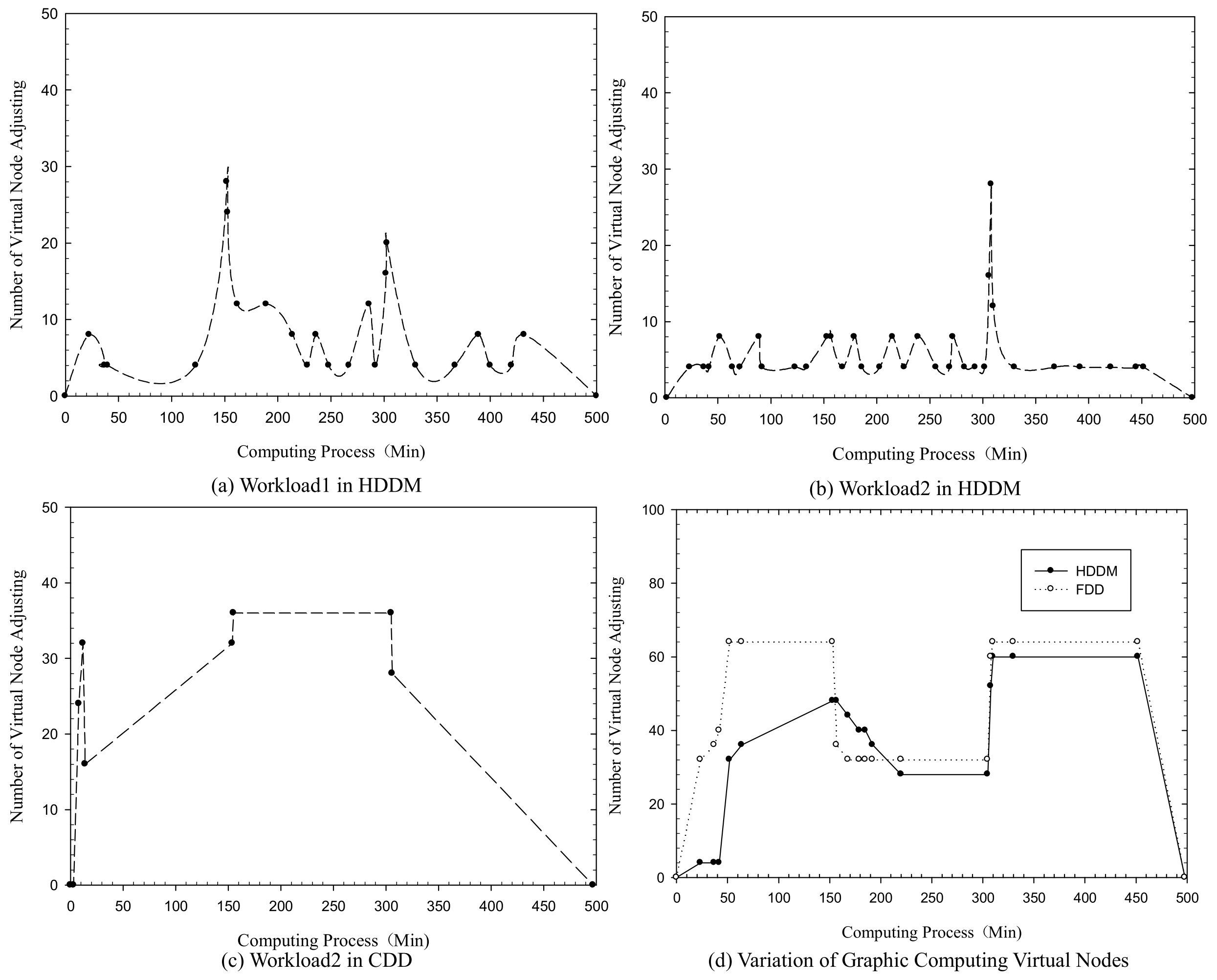


FIGURE 9. Comparison of two artificial workloads for HDDM.

resource allocation oscillation. This is a advantage compared with CDD and FDD.

C. PERFORMANCE ANALYSIS

This subsection compares and analyzes performance of SDM, CDD, FDD and HDDM. We used the workload1 to compare performance rather than workload2. This is because that workload2 is constructed deliberately in order to validate the advantage of HDDM in resource allocation oscillation. The average completion time of jobs is chosen as performance metric since it can represent the comprehensive evaluation of cloud systems or Large-scale virtual clusters, such as, the ability of responding task requests to users, the transmitting ability of data and virtual node deployment overhead. The experimental results are shown as Table 5. It can be seen that the performance of virtual cluster is different by using four different deployment models. Each model has been run ten times to eliminate performance deviation. The mathematical expectation ε and standard deviation σ of job completion time are collected and listed as follows.

TABLE 5. Performance comparison of different deployment scheme.

	SDM		CDD		FDD		HDDM	
	ε	σ	ε	σ	ε	σ	ε	σ
Average Time(s)	1593	127.4	1557	77.8	1553	48.1	1468	14.9

The average completion time of HDDM is smallest among these approaches. Additionally, the smallest standard deviation value represents that HDDM has best stability. According to the results of experiments, HDDM has a better effect of virtual node deploying and thus to improve the resource utilization of virtual cluster effectively. This is because HDDM can dynamically build a more optimized execution environment for jobs in real time and reduce the waiting time of jobs. Furthermore, HDDM can provide more virtual nodes for the running jobs from the global and local optimal perspective to ensure performance and stability. CDD and FDD are better than SDM since the dynamical deployment mechanism has the ability of optimization for job performance. The standard deviation value of FDD is smaller than that of CDD. This illustrates that the fine-grained deployment from the local optimal perspective is more stability than CDD from the global perspective. However, the deployment overhead of FDD is slightly greater than CDD. CDD saves the deployment time and start time of virtual node since it deploys more virtual nodes than FDD in a short period. To sum up, the conclusion that HDDM has better efficiency and stability.

D. RESOURCE UTILIZATION ANALYSIS

This subsection compares the resource utilization. The approach of collecting resource utilization is follows: the resource utilization is monitored by software CAdvisor, which is able to capture performance data based on Docker API. CAdvisor can collect CPU, memory, disk and Network I/O utilization for two minutes. Then influxDB is used to save the monitored data of all virtual nodes and count the average for different subclusters. First, CPU utilization of virtual node is an important indicator of resource utilization. Note that the original CPU utilization needs to be dealt with since this value may exceed 100% when multi-processes are running. The CPU utilization rate is defined as the time rate of occupied by processes. Assume that the sampling period is 1000*ms*. When only one process is running on a CPU core, this process takes up 250*ms* of CPU. So the CPU utilization rate in this sampling period is 25%. When four processes are running on two CPU cores, each process runs for 250*ms*. The original CPU utilization rate captured by CAdvisor is 100%. However the final value is 50% since these processes are running on two cores.

Figure 10 provides the four different deployment approaches comparison with workloads1. Compare with other approaches, HDDM achieves the best CPU utilization rate in subclusters. Additionally, FDD is relatively better than CDD and SDM. This means that FDD has the ability of accurate deploying to improve resource utilization. The CPU utilization rate of batch computing, memory computing and stream computing subclusters are similar. The CPU utilization rate of graphic computing subcluster is smaller than the others due to the less assigned jobs.

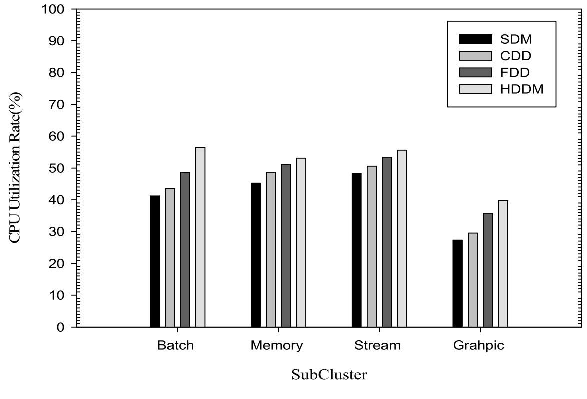


FIGURE 10. Comparison of CPU utilization for four approaches.

As shown in Fig. 11 and Fig. 12, HDDM still achieves the best memory and network utilization rate in subclusters. In Fig. 11, the memory utilization rates of memory computing subcluster reach the highest value. And the Network utilization of stream computing subcluster is maximum value in Fig. 12. This is due to the fact that the subcluster needs more specific resources for specific computing mode. These results illustrates that HDDM can improve the resource utilization for different jobs including CPU-intensive jobs, memory-intensive jobs and Network-intensive jobs.

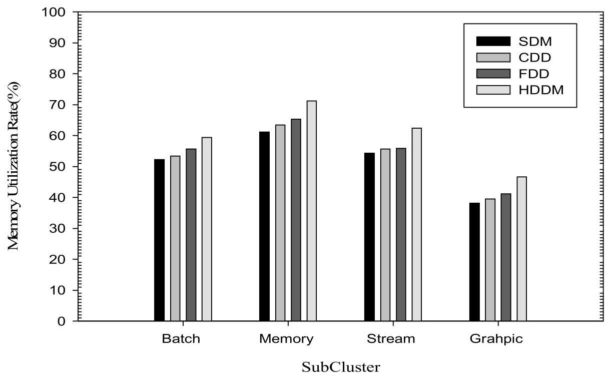


FIGURE 11. Comparison of memory utilization for four approaches.

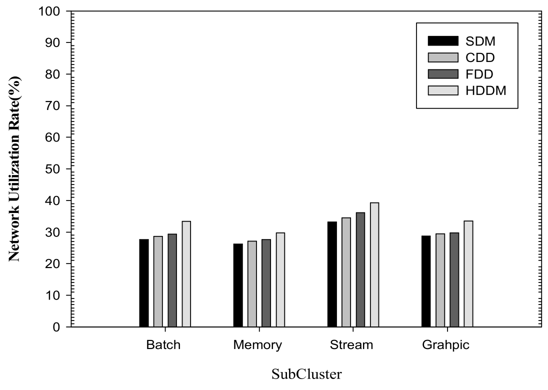


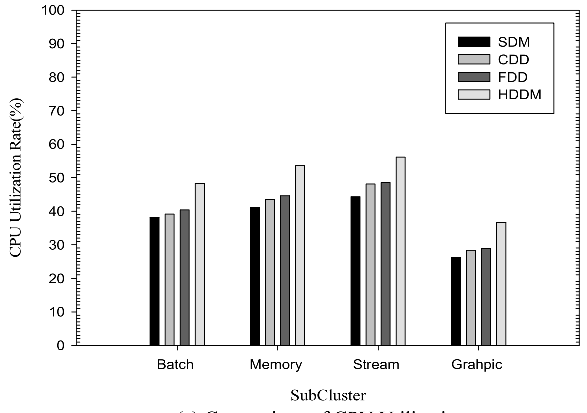
FIGURE 12. Comparison of network utilization for four approaches.

E. REAL-WORLD WORKLOADS ANALYSIS

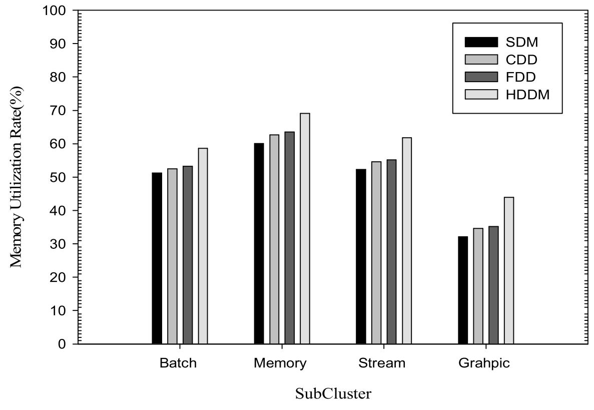
In this subsection the real-world dataset was used to evaluate performance of HDDM. This real-world workload is from real-time computing by taxi trajectories datasets generated by 33000 taxis over a period of six months [29], [30]. The taxi trajectory is a sequence of GPS points pertaining to a trip. Each point consists of a longitude, latitude and timestamp. Here the complex jobs are trajectory matching and congestion road identification. Trajectory matching is a typical data-intensive computing job that includes batch computing tasks and memory computing tasks. Congestion road identification that belongs to traffic flow analysis is an important application for navigation system and autonomous driver. This computing job is to identify the congestion road section in map by analyzing trajectory data. And it includes stream computing tasks, memory computing tasks and graphic computing tasks.

The experiments stimulate a large number of users to submit jobs with different data size. For example, job1 is trajectory matching job with 500MB trajectory data and job 2 is trajectory matching with 20GB trajectory data. Job1 and job2 are regarded as different complex jobs. So the workloads generated from the real-world large-scale data set are used to evaluate HDDM. There are about 2000 submitted jobs and the computing process lasts for about five hours.

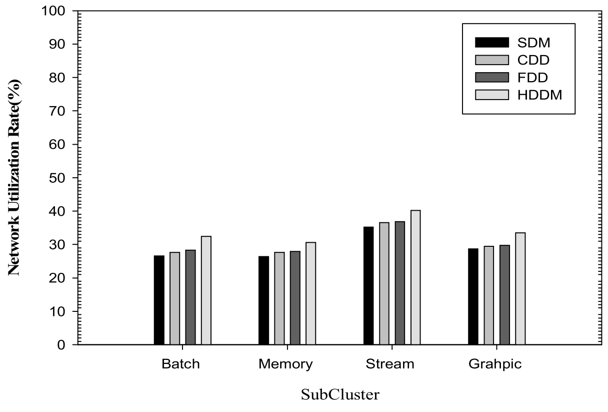
As shown in Fig. 13(a),(b),(c), the resource utilization is also optimized by HDDM to the real-world workload. On the other hand, the computing time of SDM is 3017.1 minutes. And the computing time of HDDM is just 2947.7 minutes. To the same workloads, HDDM improve the performance by 2.3 percent compared with SDM. This means that HDDM can schedule better resource for jobs to improve performance. And the performance improvement is greater than the deployment overhead.



(a) Comparison of CPU Utilization



(b) Comparison of Memory Utilization



(c) Comparison of Network Utilization

FIGURE 13. Comparison of resource utilization with real-world workloads for four approaches.

V. CONCLUSION AND FUTURE WORK

Although a unified large-scale data process platform is convenient and easy to use for large-scale data computing jobs, it ignores the differentiated resource requirements of various tasks with different computing modes. In this paper, we tackle a practical yet challenging issue of automatic deploying containers in virtual cluster. We have proposed and developed a self-adaptive cluster-level dynamic deploying approach, which automatically find the optimal settings of container configuration for different subclusters running with different computing modes. The proposed model has combined the global and the local optimization idea to pick out the optimal virtual node set in order to achieve the resource provision balance from the long-term perspective. Utilizing the coarse-grained deploying strategy with prediction-driven is to reduce deployment overhead and adapt to large resource demand adjusting. On the contrary, the fine-grained deploying strategy with event-triggering enable that this model has the capability of non-periodic and random deployment to improve the accuracy of virtual resource adjustment. Experimental results show that the proposed hybrid deployment model improves the average job completion time by 7.84, 5.71 and 5.47 percent and improve the virtual CPU utilization by 25.9, 18.6 and 7.9 percent compared to static deployment, coarse-grained deployment and fine-grained deployment, respectively.

Our method can be integrated into Swarm with some additional effort. Besides, the proposed model ignores the performance loss of container image files from local disks and the type of virtual networks in virtual cluster. A large scale of simulation experiments and prototype experiments are one of our next research works. We will attempt to implement this hybrid deploying model in a real cloud computing environment and evaluate its performance and efficiency.

ACKNOWLEDGMENT

This article was presented in part at the INFOCOM 2020 Workshops - ICCN. The authors would like to thank the comments of the reviewers.

REFERENCES

[1] C. de Alfonso, A. Calatrava, and G. Moltó, "Container-based virtual elastic clusters," *J. Syst. Softw.*, vol. 127, pp. 1–11, May 2017.
[2] D. Cheng, J. Rao, Y. Guo, C. Jiang, and X. Zhou, "Improving performance of heterogeneous MapReduce clusters with adaptive task tuning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 774–785, Mar. 2017.
[3] Q. Liu, W. Zheng, M. Zhang, Y. Wang, and K. Yu, "Docker-based automatic deployment for nuclear fusion experimental data archive cluster," *IEEE Trans. Plasma Sci.*, vol. 46, no. 5, pp. 1281–1284, May 2018.
[4] Z. Kozhirbayev and R. O. Sinnott, "A performance comparison of container-based technologies for the cloud," *Future Gener. Comput. Syst.*, vol. 68, pp. 175–182, Mar. 2017.

[5] E. Truyen, M. Bruzek, D. Van Landuyt, B. Lagaisse, and W. Joosen, "Evaluation of container orchestration systems for deploying and managing NoSQL database clusters," in *Proc. IEEE 11th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2018, pp. 468–475.

[6] A. J. Younge, K. Pedretti, R. E. Grant, and R. Brightwell, "A tale of two systems: Using containers to deploy HPC applications on supercomputers and clouds," in *Proc. IEEE Int. Conf. Cloud Comput. Technol. Sci. (CloudCom)*, Dec. 2017, pp. 74–81.

[7] M. GroBmann and C. Klug, "Monitoring container services at the network edge," in *Proc. 29th Int. Teletraffic Congr. (ITC)*, Sep. 2017, pp. 130–133.

[8] A. Ahmed and G. Pierre, "Docker container deployment in fog computing infrastructures," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2018, pp. 2–8.

[9] I. Mavridis and H. Karatza, "Performance and overhead study of containers running on top of virtual machines," in *Proc. IEEE 19th Conf. Bus. Informat. (CBI)*, Jul. 2017, pp. 32–38.

[10] H. Zeng, B. Wang, W. Deng, and J. Tang, "A prototype for analyzing the Internet routing system based on spark and docker," in *Proc. IEEE 7th Int. Symp. Cloud Service Comput. (SC2)*, Nov. 2017, pp. 267–270.

[11] Z. Lei, H. Du, S. Chen, C. Zhu, and X. Liu, "DCSPARK: Virtualizing spark using docker containers," in *Proc. Int. Conf. Audio, Lang. Image Process. (ICALIP)*, Jul. 2016, pp. 13–18.

[12] B. Ruan, H. Huang, S. Wu, and H. Jin, "A performance study of containers in cloud environment," in *Proc. Asia–Pacific Services Comput. Conf.*, 2016, pp. 343–356.

[13] K. Ye and Y. Ji, "Performance tuning and modeling for big data applications in docker containers," in *Proc. Int. Conf. Netw., Archit., Storage (NAS)*, Aug. 2017, pp. 1–6.

[14] N. Naik, "Docker container-based big data processing system in multiple clouds for everyone," in *Proc. IEEE Int. Syst. Eng. Symp. (ISSE)*, Oct. 2017, pp. 1–7.

[15] J. Kovács, P. Kacsuk, and M. Emődi, "Deploying docker swarm cluster on hybrid clouds using occopus," *Adv. Eng. Softw.*, vol. 125, pp. 136–145, Nov. 2018, doi: 10.1016/j.advengsoft.2018.08.001.

[16] D. Zhang, B.-H. Yan, Z. Feng, C. Zhang, and Y.-X. Wang, "Container oriented job scheduling using linear programming model," in *Proc. 3rd Int. Conf. Inf. Manage. (ICIM)*, Apr. 2017, pp. 174–180.

[17] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–6.

[18] B. Liu, P. Li, W. Lin, N. Shu, Y. Li, and V. Chang, "A new container scheduling algorithm based on multi-objective optimization," *Soft Comput.*, vol. 22, no. 23, pp. 7741–7752, Dec. 2018.

[19] J. Zhao, K. Yang, X. Wei, Y. Ding, L. Hu, and G. Xu, "A heuristic clustering-based task deployment approach for load balancing using Bayes theorem in cloud environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 305–316, Feb. 2016.

[20] O. Adam, Y. C. Lee, and A. Y. Zomaya, "Stochastic resource provisioning for containerized multi-tier Web services in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2060–2073, Jul. 2017.

[21] Q. Song, "On the weight convergence of elman networks," *IEEE Trans. Neural Netw.*, vol. 21, no. 3, pp. 463–480, Mar. 2010.

[22] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, "Autonomic resource management in virtualized data centers using fuzzy logic-based approaches," *Cluster Comput.*, vol. 11, no. 3, pp. 213–227, Sep. 2008.

[23] R. Chandra, "Competition and collaboration in cooperative coevolution of elman recurrent neural networks for time-series prediction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3123–3135, Mar. 2015.

[24] C.-M. Lin and E.-A. Boldbaatar, "Fault accommodation control for a biped robot using a recurrent wavelet elman neural network," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2882–2893, Dec. 2017.

[25] W. Z. Sun and J. S. Wang, "Elman neural network soft-sensor model of conversion velocity in polymerization process optimized by chaos whale optimization algorithm," *IEEE Access*, vol. 5, pp. 13062–13076, 2017.

[26] J. Bhimani, Z. Yang, N. Mi, J. Yang, Q. Xu, M. Awasthi, R. Pandurangan, and V. Balakrishnan, "Docker container scheduler for I/O intensive applications running on NVMe SSDs," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 4, no. 3, pp. 313–325, Feb. 2018.

[27] D. Gedia and L. Perigo, "Performance evaluation of SDN-VNF in virtual machine and container," in *Proc. IEEE Conf. Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, Nov. 2018, pp. 1–7.

[28] J. Liu, Y. Liang, and N. Ansari, "Spark-based large-scale matrix inversion for big data processing," *IEEE Access*, vol. 4, pp. 2166–2176, 2016.

[29] Y.-L. Hsueh and H.-C. Chen, "Map matching for low-sampling-rate GPS trajectories by exploring real-time moving directions," *Inf. Sci.*, vols. 433–434, pp. 55–69, Apr. 2018.

[30] H. Wang and Y. Cao, "An energy efficiency optimization and control model for Hadoop clusters," *IEEE Access*, vol. 7, pp. 40534–40549, 2019.

[31] M. Caballer, C. de Alfonso, G. Moltó, E. Romero, I. Blanquer, and A. García, "CodeCloud: A platform to enable execution of programming models on the clouds," *J. Syst. Softw.*, vol. 93, pp. 187–198, Jul. 2014.

[32] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "CloudScale: Elastic resource scaling for multi-tenant cloud systems," in *Proc. 2nd ACM Symp. Cloud Comput. (SOCC)*, 2011, pp. 1–14.

YUNPENG CAO was born in Yinan, Shandong, China, in 1967. He received the bachelor's degree from Nankai University, in 1989, and the master's degree from the Shandong University of Science and Technology, in 2005. He is currently a Vice Professor with Linyi University. His research interests include network security, network computing, GPGPU, parallel computing, and intelligence control simulation.

HAIFENG WANG was born in Linyi, Shandong, China, in 1976. He received the bachelor's and master's degrees from the China University of Petroleum, in 1998 and 2005, respectively, and the Ph.D. degree in computer science from the University of Shanghai for Science and Technology, in 2012. He is currently a Professor with Linyi University. His research interests include network security, GPU computing, network computing, GPGPU, high-performance computing, and intelligence control systems.