MarVeLScaler: A Multi-View Learning based Auto-Scaling System for MapReduce

Yi Li, Fangming Liu*, *Senior Member, IEEE,* Qiong Chen, Yibing Sheng, Miao Zhao, *Member, IEEE,* and Jianping Wang, *Member, IEEE*

Abstract—To promote cloud computing from current pay-per-request model to truly pay-per-use, tenants are crying for automatic tools to auto-estimate the amount of resources for MapReduce jobs. Such tools call for accurately quantifying the relationship among workload, resources and completion time. Various prediction models have been proposed. However, none of these models takes virtual machines' (VMs) performance variance during a job's execution into consideration, leading to underestimate the required resources and exceed the job's deadline. To address this problem, we propose a multi-view deep learning model to capture real-time performance variance and automatically scale out the cloud cluster whenever necessary. We implement *MarVeLScaler*, a prototype system including two useful modules, namely, *Scale Estimator* and *Scale Controller*. Scale Estimator preliminarily estimates the required cluster size for a MapReduce job with given a concrete workload and deadline. During the runtime, Scale Controller adjusts the scale of the cluster according to its real-time running status to guarantee the job finished on time. We evaluate the performance of MarVeLScaler based on Hadoop in Alibaba Cloud. Experiments show that MarVeLScaler can provide 98.4% accuracy of prediction in determining initial cluster size, and save 30.8% of expense while still guaranteeing similar performance compared with the state-of-the-art methods.

Index Terms—Cloud Computing, Auto-scaling, Multi-view Neural Network.

1 Introduction

The common practice of current cloud computing [1] is that a tenant tells the cloud service provider the amount of requested resources (i.e., the number of VMs and their configurations), then the cloud service provider accordingly allocates these resources [2] for the tenant, where virtualization technology is used to improve resource utilization. Tenants who want to finish their computing job, such as MapReduce, by a specific deadline can benefit from this practice since they can obtain plenty of computing resources without purchasing these expensive servers. However, this practice also makes tenants have to estimate their own demand for resources. From the perspective of tenants, it is hard for them to precisely estimate the most appropriate amount of resources to meet the deadline due to the following two reasons. Firstly, tenants lack the knowledge (i.e., the quantitative relationship among the workload, the computing resources and the completion time of a job) of estimating the requested resources to start VMs. Secondly, even though a tenant can make a good estimation in a bare-metal server environment, such estimation may not be sufficient in a virtual environment, because the performance of a VM is usually unstable in a public cloud [3].

Someone may suggest that the above problems can be addressed by existing auto-scaling systems. That is, tenants can always start with a small amount of resources and then the auto-scaling system can add up more resources in the later stage. Unfortunately, most current auto-scaling systems in public clouds cannot work well on MapReduce jobs. These systems are usually rule-based, which require tenants to configure when and how much to scale. For example, a tenant can set to scale out one VM when the CPU utilization is higher than a certain threshold [4] [5] [6]. However, when a MapReduce job is in execution, the CPU utilization will always maintain at a full level, which makes it hard to decide the auto-scaling threshold. In fact, other commonly used rules (e.g., memory utilization) are not good indicators of triggering the scaling operation as well [7] [8]. First, threshold based methods usually require tenants to configure these thresholds, which requires empirical knowledge [7]. Second, static and predefined thresholds may not work well when the workload changes. As a result, tenants have to adjust these thresholds according to the workload, which is time consuming [8].

Thus, we aim to design a prediction based auto-scaling system for MapReduce jobs, but it faces the following two challenges. The first challenge is that building such an auto-scaling system without unpractical assumptions is difficult

- *This work was supported in part by the National Key Research & Development (R&D) Plan under grant 2017YFB1001703, in part by NSFC under Grant 61722206 and 61761136014 (and 392046569 of NSFC-DFG) and 61520106005, in part by the Fundamental Research Funds for the Central Universities under Grant 2017KFKJXX009 and 3004210116, in part by the National Program for Support of Top-notch Young Professionals in National Program for Special Support of Eminent Professionals, in part by Hong Kong Research Grant Council under CRF C7036-15G and NSFC-Guangdong Joint Fund under project U1501254.*
- *Y. Li, F. Liu and Q. Chen are with the National Engineering Research Center for Big Data Technology and System, the Services Computing Technology and System Lab, Cluster and Grid Computing Lab in the School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan, 430074, China. E-mail: {m201672796, fmliu, qiongchen}@hust.edu.cn. The corresponding author is Fangming Liu.*
- *Y. Sheng is with the Department of Computer Science, University of Toronto, 27 King's College Circle, Toronto, ON, CA. E-mail: yibing.sheng@mail.utoronto.ca.*
- *M. Zhao is with the Department of Computing, Hong Kong Polytechnic University, Kowloon, Hong Kong. E-mail: mzhao.ny@gmail.com.*
- *J. Wang is with the Department of Computer Science, City University of Hong Kong, Kowloon 8523, Hong Kong. E-mail: jianwang@cityu.edu.hk.*

because these assumptions are adopted to simplify the construction of the prediction model. Usually, the prediction of a MapReduce job's completion time requires detailed analyses of its running procedure. However, a MapReduce job contains several phases. Each phase requires different resources and depends on each other. It is difficult to directly characterize the relationship among workload, resources and completion time for all phases of the job. Thus, previous studies [9] [10] [11] [12] usually assume that the Map and Reduce phases of a MapReduce job do not overlap with each other, which means that the whole procedure is decomposed into two completely separate sub-procedures and the total completion time will be the sum of the run-time in both phases. However, this assumption is not practical in production environments, because this will lead to the waste of resources. For example, SlowStart [13] is a running setting parameter in Hadoop which decides the overlap degree of the Map and Reduce phase. This parameter will seldom be set to 1 (i.e., totally no overlap). As a result, the prediction models in [9] [10] [11] [12] cannot predict well in this situation.

The second challenge is that the performance degradation of VMs badly cripples the effectiveness of existing methods [14] [15] [16] [17]. These methods are only based on some offline features (e.g., resource features and job-related parameters). Directly applying these methods can easily result in significant malfunction when VMs performance degradation happens. Specifically, to guarantee finishing the job by deadline usually requires the construction of a prediction model to figure out the relationship among needed computing resources, workload and completion time. However, this relationship varies when VMs performance degrades, which deviates the prediction from the expected result and even lead to wrong scaling decisions. As to be shown in the following Section 2, given the same workload and VM configurations for a MapReduce job, if we run the job multiple times on the same cloud cluster, the completion time over multiple runs varies significantly and break previous relationships with computing resources and workload [18]. Therefore, to handle the performance degradation of VMs is extremely critical in the auto-scaling system design.

To address the above challenges, we present *MarVeLScaler*, a two-stage auto-scaling system designed for tenants. In the first stage, MarVeLScaler provides an accurate and straightforward working module, referred to as *Scale Estimator*, which estimates the required number of VMs under different configurations for a MapReduce job with given its workload and deadline. Specifically, Scale Estimator applies a regression model that predicts the total processing time directly based on resource features (i.e., the configuration of a VM and the number of VMs) and workload. Thus, instead of analyzing the running procedure of a MapReduce job, our idea is to leverage a regression model to capture the quantitative relationship among the workload, the computing resources and the completion time of a job. Thus, our model avoids assuming that the Map and Reduce phases of a MapReduce job do not overlap with each other. Besides, the output of the Scale Estimator can be directly applied to generate a detailed resource purchase list for tenants to set up the cloud cluster for performing MapReduce jobs.

In the second stage, MarVeLScaler offers another working module to guarantee the job finished by the deadline, referred to as *Scale Controller*, which fine-tunes the resource estimation and gives online scaling recommendations in terms of when and how much to scale. As mentioned before, although some offline features are good indicators of predicting the completion time, the performance degradation of VMs will weaken the effectiveness of these features. To deal with this problem, we propose using online features, such as running speed features and resource utilization features, to solve the degradation problem because they can timely indicate the job running status, and Scale Controller can accordingly adjust the prediction whenever necessary. Specifically, Scale Controller applies a Deep Canonically Correlated AutoEncoder (DCCAE) [19] structure to refine a high-level representation from the raw multi-view features (i.e., offline and online features), where this structure is suitable to simultaneously learn different views of features and maximize the correlations of features across views. Thus, the predictions using this presentation would be more accurate and sufficient to make scaling decisions.

In a brief summary, MarVeLScaler jointly using Scale Estimator and Scale Controller can effectively help tenants find the most cost-saving strategy of guaranteeing MapReduce jobs to finish on time in public clouds. The main contributions of this paper are summarized as follows:

- We develop Scale Estimator to help tenants choose the most cost-saving amount of resources to set up the cloud cluster for performing MapReduce jobs. Specifically, it applies a regression model to provide precise predictions (i.e., 98.4%) with only a limited number of training samples.
- We develop Scale Controller to guarantee a MapReduce job finished on time in public clouds by analyzing its real-time running status. Specifically, it applies a multi-view deep learning model to capture real-time performance variance of VMs and automatically scale out the cloud cluster whenever necessary.
- We implement MarVeLScaler, a prototype auto-scaling system including the above two practical modules (i.e., Scale Estimator and Scale Controller) based on Hadoop in Alibaba cloud. The experiment results show that MarVeLScaler reduces 30.8% of expenses while still guaranteeing similar performance compared with state-of-the-art methods.

2 Motivation

In this section, we present the motivations to build a two-stage auto-scaling system for performing MapReduce jobs in public clouds.

2.1 Initial Resource Decision

Given the workload and preferred deadline, it is not easy for a tenant to determine the best choice of initial resources. Typically, a MapReduce job can be finished in a similar time duration under different resource settings of VM configurations and number of VMs, and each setting has different costs. Specifically, we conduct experiments on

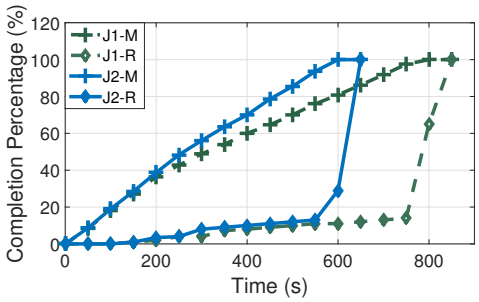


Fig. 1: The detailed running progress of 20 VMs and 200 GB workload in the best and worst conditions.

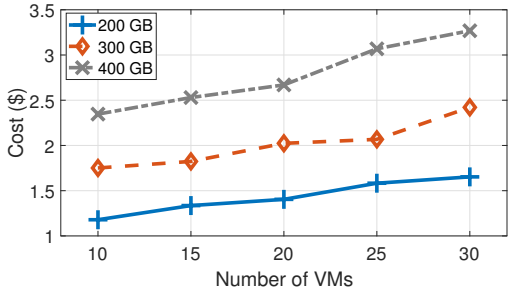


Fig. 2: The cost ($) under different numbers of VMs with the same specification for a given workload.

TABLE 1: Different resource settings may lead to a similar completion time for a 200 GB WordCount job.

No. of VMs	Mem (GB)	Storage	CPU	Time (s)	Cost ($)
28	16	SSD	Exclusive	449	1.09
32	8	HDD	Exclusive	440	0.99
36	16	SSD	Shared	449	1.62
40	8	SSD	Shared	456	1.40

TABLE 2: The same workload and resources with different completion time over multiple runs.

No. of VMs	Workload (GB)	Min Time (s)	Max Time (s)
20	200	533	746
30	450	789	903
40	500	713	928

Alibaba cloud by running a WordCount job with 200 GB workload under different VM configurations [1]. The results are shown in Table 1. We can see that for a given workload, there are multiple options of resources where each job can be finished in a similar time period, which indicates that computing resources may accelerate the running procedure of a specific kind of workload in varying degrees. On one hand, comparing the resource settings in row 1 and row 3, exclusively using CPU [2] significantly accelerates the computation process of a WordCount job and reduce the number of used VMs. On the other hand, from row 1 and row 2, introducing more VMs seems more cost-effective, instead of using larger memory and expensive storage, such as SSD, because WordCount workload is CPU intensive. Thus, it is necessary to develop an auto-estimator to provide a recommendation on the least payment for tenants.

2.2 Performance Variance

Even with a good choice of initial resources, a job may still not be finished within the expected deadline because of performance variance of VMs in a cloud environment. To demonstrate the performance variance, we conduct experiments in Alibaba cloud by comparing the completion time of WordCount jobs over multiple runs under the same settings. Specifically, we apply the same cloud cluster to run the same workload for 10 times. As shown in Table 2, the minimum and maximum completion time vary significantly, which is due to the sharing characteristic of the virtual environments. For further study of the performance variance, we conduct experiments to profile the MapReduce jobs to figure out how the running speed changes during the runtime. As shown in Fig. 1 where M stands for Map phase and R for Reduce phase, Job 1 (J1) suffers the performance degradation while Job 2 (J2) does not. At around 200 s, the speed of Job 1 decreases while the speed of Job 2 maintains the same. Thus, it is crucial to capture the moments of performance degradation (i.e., slower running speed than expected) and timely make new resource estimation in order to meet the deadline.

2.3 Disapproval for Additional Resources

Someone may suggest that according to pay-per-use charge policy if more VMs are assigned initially, it takes less processing time to finish the job, and the payment may be less than that with fewer VMs. If this is true, it may be unnecessary to do an online resource estimation. To dispel this concern, we conduct experiments in Alibaba cloud by comparing the cost under different numbers of VMs for the same MapReduce job, where all the VMs initially use the same specification and configuration. We can see in Fig. 2, as the number of VMs increases, the cost of finishing the MapReduce job also gradually increases, though more VMs lead to shorter processing time. Thus, it is more beneficial for tenants by starting a job with barely-meeting-deadline resources and conducting necessary scaling through online prediction.

2.4 Motivation of Multi-view Neural Network

As we can see from the above, the auto-scaling system is important for cloud tenants to guarantee their MapReduce jobs to finish on time while keeping the payment at a minimum. This usually consists of two stages: 1) initial resource decision, and 2) runtime dynamic resource adjustment. However, it is extremely difficult to achieve this objective in practice due to the following reasons: 1) Overprovisioning resources and undershooting the deadline is wasteful; 2) Exact performance prediction in the presence of failures and dynamic events is difficult; 3) Heterogeneity plus the diminishing return of extra parallelism make performance nonlinear in the resources committed to a job. Hence, the key

1. All the VMs in this paper are equipped with 4 Intel Xeon E5-2682 v4 2.5GHz CPU.

2. Alibaba Cloud provides entry-level users with the shared CPU service which is cheaper and has limited computing ability.

to achieving the above objective lies in preliminarily start MapReduce jobs with barely-meeting-deadline resources, and dynamically scale-out when necessary through online prediction of MapReduce job's completion time.

As a remedy for such a challenge, our idea is to leverage machine learning techniques to estimate and dynamically control the exact resources required to finish a MapReduce job by a deadline. Specifically, we apply a multi-view deep learning model to overcome the deadline violation problem caused by VM performance degradation. In contrast, most previous studies of auto-scaling on public cloud usually do not take VM performance variations into account. The multi-view deep learning model is good at exploring highly non-linear relationships that exist among low-level features across different groups [20] [21]. Taking this advantage, our proposed system is able to comprehensively capture the running status of a MapReduce job by extracting key information from three different groups of features (i.e., computing resources, resource utilization, and processing speed). For interested readers, a more detailed implementation of the multi-view neural network is also available in Section 3.3.2.

3 System Overview and Design

In this section, we first give an overview of MarVeLScaler, and then introduce the detailed design of Scale Estimator and Scale Controller, respectively.

3.1 Overview of MarVeLScaler

Fig. 3 shows the workflow of MarVeLScaler. Our system consists of two modules, namely, Scale Estimator (SE) and Scale Controller (SC). Before MarVeLScaler starts to work, historical data (i,e., executed jobs, and their logs) are collected for fitting the working unit inside Scale Estimator, i.e., *Allocator*, and training the corresponding working unit inside Scale Controller, i.e., *Predictor*. Upon receiving the workload and deadline of a MapReduce job from a tenant, *Allocator* generates a detailed resource purchase list. Then, the tenant picks the most cost-saving option and requests providers to set up the cloud cluster with corresponding computing resources, like the number of VMs and configurations. When the job is in execution, the other working unit inside Scale Controller, i.e., *Monitor*, constantly collects online data (i.e., running speed and resource utilization) from the cluster to monitor the job's running status. These online data are processed into features as the input of *Predictor*. As soon as the Predictor receives these data, it will predict the completion time. If Predictor finds that the completion time goes beyond the deadline, it will automatically scale additional VMs into the cloud cluster to ensure that the completion time is shortened to right before the deadline. In a brief summary, MarVeLScaler jointly using Scale Estimator and Scale Controller can effectively help tenants find the most cost-saving strategy of guaranteeing MapReduce jobs to finish on time in public clouds.

3.2 Scale Estimator Design

As mentioned before, it is crucial for Scale Estimator design to have the ability to predict completion time based on

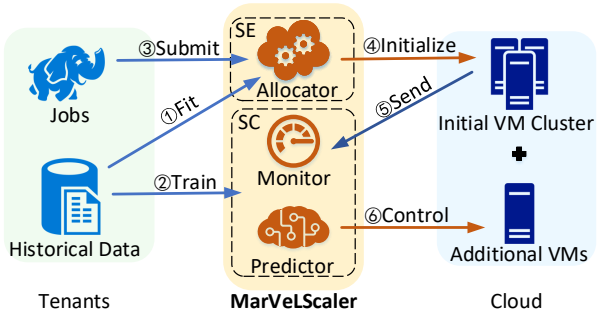


Fig. 3: The workflow of MarVeLScaler.

TABLE 3: Resource features.

Symbol	Description
w	The total amount of workload to be processed
n	The number of VMs in the cluster
R	The volume of the memory
H	The type of the storage medium
U	The way in which the resources are used
G	The architecture of CPU

features of the workload and computing resources shown in Table 3. Our idea is to leverage machine learning techniques to capture the quantitative relationship among workload, the computing resources and the completion time of a job. Specifically, we apply a regression model due to the following two reasons.

- It can provide better prediction accuracy when the number of training samples is limited. Neural network is prone to cause an over-fitting problem when the number of input features has the same quantitative level as the training data [22]. In this situation, neural network will perform well on the training data-set but cannot perform beyond the training set. However, regression model can solve this problem by restricting the complexity of the formulas [23]. As a result, when the number of training samples is limited, neural network will be over-fitting, while regression model will still provide satisfactory predictions with proper formulas.
- We leave the guarantee on the deadline to Scale Controller so that Scale Estimator can be offline. The prediction errors of the regression model will be solved by dynamic adjustments offered by Scale Controller.

3.2.1 Regression Model Design

The completion time of a MapReduce job is determined by the computing resources and its workload, we formulate the relationship among these parameters as follows:

$$T_{estimate} = \mathscr{F}(w, n, H, U, R, G) \quad (1)$$

where $T_{estimate}$ is the estimated completion time of a MapReduce job and $\mathscr{F}(\cdot)$ is a function mapping the resources and workload to $T_{estimate}$. The other notations of parameters used in Eq.1 are defined in Table 3. Then, in order to derive a precise formulation of $\mathscr{F}(\cdot)$, we first figure out the linear correlation between these features and the completion time by using Pearson correlation coefficient

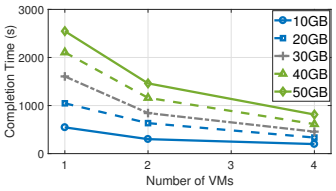


Fig. 4: The completion time as a function of the number of VMs.

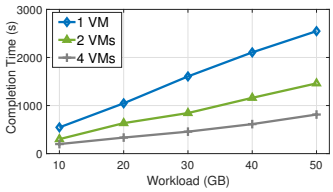


Fig. 5: The completion time as a function of the workload.

TABLE 4: Pearson correlation coefficients.

Symbol	Value	Symbol	Value
w	0.7002	U	0.1893
n	-0.66	H	0.1968
R	-0.2184	G	-0.2184

(PCC), which is commonly used to describe the linear correlation between two variables in statistics. Specifically, we calculate the PCC between each feature and its corresponding completion time, by letting the covariance of these two variables divided by the product of their standard deviations [24]. PCC must be between -1 and 1, where 1 means a completely positive linear correlation, 0 means non-linear correlation, and -1 means a completely negative linear correlation. The bigger an absolute value of PCC is, the greater the impact on the completion time a feature has.

To obtain the PCC of each feature in Table 3, we collect the data from 80 samples, and each sample represents the result of executing a WordCount job. As shown in Table 4, the total amount of workload to be processed, w, and the number of workers in the cluster, n, each have a relatively strong correlation with the completion time. It means that these two have a bigger effect than other features do. Thus, we separately analyze the relationship between these two features and the completion time to formulate our $\mathscr{F}(\cdot)$.

As shown in Fig. 4 and Fig. 5, Consistent with our intuition, the completion time decreases as the number of VMs increases, but increases as the workload increases. Thus, we can conclude that the completion time is negatively related to the number of VMs in the job, but positively related to workload. As for other features, we assume that they have a linear relationship with the completion time and do not interact with each other. Thus, the estimated completion time is formulated as follows:

$$T_{estimate} = b_0 + \frac{b_1 w}{n(b_2 + b_3 H + b_4 U + b_5 R + b_6 G)} \quad (2)$$

where $b_0, ..., b_6$ denote the coefficients need to be fitted. Inside the numerator, w reflects the total amount of workload of a MapReduce job. As for the denominator, let the linear combination of H, U, R and G represent the computing ability of one VM. Using a single computing ability of one VM to multiply the total number of VMs gives us the overall computing ability of a cluster, which is represented by the denominator. In total, let the entire workload be divided by the overall computing speed, then we get the total time of completing one job along with the bias, b_0.

3.3 Scale Controller Design

As mentioned before, most previous studies of auto-scaling on public clouds usually do not take VMs performance variations into account which lead to wrong scaling decisions and exceed the deadline. To overcome the deadline violation problem caused by VM performance degradation, our idea is to apply a multi-view deep learning model, which is good at exploring highly non-linear relationships that exist among low-level features across different groups. In the following, we briefly introduce the domain-assisted feature engineering.

3.3.1 *Domain-assisted Feature Engineering*

In the auto-scaling domain, there is usually no luxury to have enormous data where a model can be trained to automatically eliminate irrelevant features. As such the first challenge is to select the proper feature set for auto-scaling decision predictions. Our feature engineering uses domain knowledge to create features relevant to the problem through measurements, which is introduced as follows.

- View 1 (V_1): The resource features are classified into this view, which is listed in Table 3. In addition, running setting parameters (e.g., SlowStart) in Hadoop is also included, which indicates that our model considers different Hadoop configurations
- View 2 (V_2): The MapReduce job running speed features are classified into this view, which is listed in Table 5. These dynamic features take the main responsibility to solve the performance variance problem in public clouds.
- View 3 (V_3): The real-time resource utilization features are classified into this view, which is listed in Table 6. These features are usually the bottleneck of performing MapReduce jobs.

Features in View 1 are selected to characterize the initial configurations of the cloud cluster and MapReduce jobs (Hadoop in our experiment). Previous studies usually construct models to characterize the non-linear relationship between these features and completion time, which only achieves satisfactory results when the performance of VMs is steady. In our model, we deal with these features with a neural network, which is more capable of building a non-linear relationship. Thus, taking all these features into account can greatly improve the prediction precision of our model.

Features in View 2 are expected to characterize the running speed of the MapReduce job, which is able to address the performance variance of VMs. As a result, when the performance of VMs degrades, the features in this view will provide better predictions about the completion time. However, it is difficult to directly characterize the job's running status. The reason behind is that Map and Reduce phases run simultaneously in a real production environment, and have totally different running speeds. Moreover, these stages will interact with each other. Specifically, once the Map tasks are finished, the computing resources will be fully used by the Reduce tasks. If we regard the processing of Map and Reduce as a whole procedure, the above-complicated process cannot be well described. Thus, the

TABLE 5: Job running speed features.

Symbol	Description
M_P	The completion percentage of Map phase
R_P	The completion percentage of Reduce phase
M_S	The running speed of Map phase
R_S	The running speed of Reduce phase
T_P	The completion percentage of the whole procedure
T_I	The average interval between two completed tasks

TABLE 6: Resource utilization features.

Symbol	Description
H_C	The CPU utilization of the header node
H_L	The one-minute average CPU load of the header node
H_M	The remaining memory of the header node
W_C	The mean CPU utilization of the worker nodes
W_L	The mean one-minute average CPU load of the worker nodes
W_M	The mean remaining memory of the worker nodes

prediction model will lose some key details about the job, and predicted completion time will be less accurate.

From the above considerations, we finally design the features of View 2 as shown in Table 5 to characterize the running speed. These features come from Map phase, Reduce phase, and the whole procedure respectively. For each phase, we use both running speed (i.e., M_S, R_S, and T_I) and completion percentage (i.e., M_P, R_P, and T_P) features. The reason why we have to introduce the completion percentage is that the corresponding running speed will not make any difference when the completion percentage in one stage is 100%. Thus, our model requires indicators to describe the importance of running speed features and better capture the interaction of Map and Reduce phases. We use T_I, the average time interval between any two completed tasks to indicate the running speed of the whole job. Using T_I instead of the running speed of the whole job is because this speed is the linear combination of the speed of each phase (i.e., M_S and R_S).

Features in View 3 are important factors to predict job processing time. For example, due to the I/O bottleneck, some CPU cores may get idle during the running time, which temporarily reduces the general computing power. We use the features in View 3 to describe the amount of resources used in the job. These features are mainly classified into two categories (i.e., header node and worker node), because they have different needs for resources. The header node is mainly responsible for the task and resource scheduling, while the worker nodes do all the computing work. The total computing resources of the cluster along with their utilization can represent the exact resource put into computing. Thus, these resource utilization features in this view would be a good complement to the resource features in View 1.

In summary, in order to conduct online scaling decisions, models based only on static features (e.g., features in Table 3) may be insufficient to catch the performance variance of VMs in public clouds. Thus, Scale Controller uses three views of features, which contain complementary information about the running job, for making more accurate predictions.

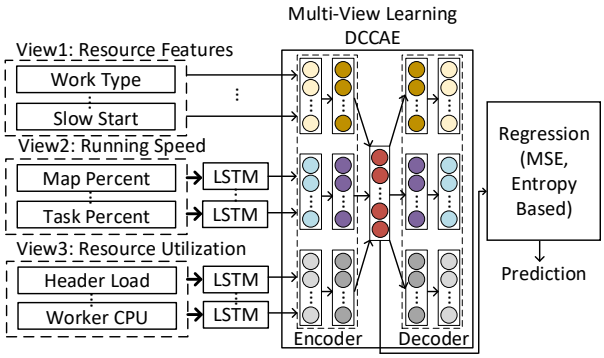


Fig. 6: The multi-view deep learning model of Scale Controller.

3.3.2 *Multi-view Deep Learning Model*

Taking all these features into account combines the advantage of resource features and running status features together. However, with more features, using regression models to model the relationship between the features and completion time becomes much more difficult. Notice that neural networks have a greater ability to construct non-linear relationships among feature. [25] proves that neural networks with sufficiently smooth activation functions are capable of accurate approximation to a function and its derivatives, which is known as the universal approximation theorem. In our system, we leverage a multi-view neural network with a sophisticated feature fusion structure. Thus, our neural network has the ability to construct an accurate enough function to describe the relationship between the given features and the completion time. Moreover, the training data would be sufficient in designing Scale Controller, because these data are extracted from every sample interval (e.g., 5 seconds in our system) during the runtime of each job. Lastly, there still exist the following two challenges for leveraging neural networks to explore highly non-linear relationships that exist among low-level features across different groups.

- Features in View 1 are non-time-dependent, while features in View 2 and View 3 are time-dependent series. Thus, it is difficult to integrate these features with different time scale together.
- Three views of features can provide a comprehensive perspective of running jobs. However, these features come from different views, and their correlation information across views is weak. This may incur significant noise in training data and influence the prediction accuracy.

A model based on these features should have the ability to be aware of performance degradation and also robust to noise. To this end, we apply a multi-view deep learning model [20] [21], which is good at exploring highly non-linear relationships that exist among low-level features across different groups. These neural networks are customized to deal with the different characteristics of different views of features. The features in our experiment can be classified into time-dependent features and non-time-dependent features. Thus, different types of neural network structures should be applied to extract high-level features from low-

level features. Then, these high-level features should be fused together for the final prediction.

Before giving a detailed introduction to the multi-view deep learning model, we introduce the following two related concepts. First, long short term memory (LSTM) [26] is a well-designed neural network, usually used to extract time dependency patterns from time series data. In this paper, LSTM is applied to extract time-dependent patterns in View 2 and View 3. The sudden decrease in speed features does not mean performance degradation, while a long term of poor running speed may indicate the performance variance. The memory cell in LSTM has the ability to make such judgments and pass proper features to the next layer.

Second, deep canonical correlation analysis (DCCA) [27], which is a deep neural network (DNN) based canonical correlation analysis (CCA), where DNNs are applied for constructing nonlinear features, and the canonical correlation between views will be maximized. DCCAE [19] is a correlation-aware auto-encoder, which concentrates on both the canonical correlation between views and the reconstruction errors of the auto-encoder.

The detailed design of our multi-view deep learning model is shown in Fig. 6. Firstly, we apply LSTM to deal with time-dependent features in View 2 and View 3. The output of the LSTM and the features in View 1 will be sent into DCCAE, which combines these features into a more compact representation. This low-noise and highly correlated representation will be the input to a regression layer and conduct the final predictions.

We formulate our objective function based on the idea of making the best use of the information captured in each view and latent patterns across views, where the auto-encoder learns a high-level presentation with a maximum lower bound on the mutual information and CCA maximizes the mutual information across views. Motivated by [19], we are to design the objective function by jointly considers the reconstruction errors of three auto-encoders and the canonical correlations across the three views.

Given the views $V_m(m = 1, 2, 3)$, the data matrices for each view can be denoted as $V = [v^1, v^2, ...v^N]$, where N is the size of the sample set. As for nonlinear relationships, f_m is the encoder network, where its network weight is denoted as W_{f_m}, and g_m is the decoder network with weight W_{g_m}. Correspondingly, we formulate our objective function as follows:

$$\min_{W_f, W_g, X, Y} \sum_{m \neq n, m=1}^{3} \sum_{n=1}^{3} -\frac{1}{N} tr(X_m^\top f_m(V_m) f_n(V_n)^\top Y_n) + \frac{\lambda}{N} \sum_{j=1}^{3} \sum_{i=1}^{N} \|v_j^i - g_j(f_j(v_j^i))\|^2 \quad (3)$$

$$s.t. \qquad X_m^\top (\frac{1}{N} f_m(V_m) f_m(V_m)^\top + r_{v_m} I) X_m = I \quad (4)$$

$$Y_n^\top (\frac{1}{N} f_n(V_n) f_n(V_n)^\top + r_{v_n} I) Y_n = I \quad (5)$$

$$x_j^\top f_m(V_m) f_n(V_n)^\top y_k = 0, \; for \; j \neq k \quad (6)$$

where $X = [x_1, x_2, ...x_L], Y = [y_1, y_2, ...y_L]$ in the first part of Eq. 3 are the optimal projection matrices, which maximize the mutual information between the projected views. The second part of Eq. 3 is the training error of an auto-encoder, which minimizes reconstruction errors between the input features and learned representation. In Eq. 4 and Eq. 5, both $r_{v_m} > 0$ and $r_{v_n} > 0$ are regularization parameters added to the diagonal of the sample auto-covariance matrices so that the regularized data can be used for estimating the covariance matrices [28]. Eq. 6 ensures that, when finding multiple pairs of projection pairs (e.g., (x_j, y_j) and (x_k, y_k)), the subsequent projection is constrained to be uncorrelated with the previous one.

Optimization Method. As shown in Fig. 6, our multi-view deep learning model is not an end-to-end network. Thus, the optimization method involves the tuning procedure of a DCCAE representation learning and the final regression. The objective function of DCCAE is in a full-batch form. However, the full-batch training algorithm [27] is computation intensive and consumes too much memory. To overcome this issue, we use a mini-batch based optimization method [29], where the gradient is estimated from a large mini-batch. After the fine-tuning of DCCAE, a representation from the middle layer of DCCAE will be sent into the regression layer, whose loss function is based on the mean square error (MSE). In general, our model first leverages a combination of LSTMs and DCCAE to generate a compact and highly correlated representation of raw low-level features [30]. Then, this representation is used to make predictions about the completion time.

3.3.3 The Choice of When and How Much to Scale

As discussed above, we are to apply a multi-view deep learning model to capture the performance variance in real time and automatically scale when it is necessary. However, once the predicted completion time is later than the deadline, Scale Controller needs to decide when a scaling operation should be triggered and how much to scale.

We should scale out the cloud cluster as early as possible due to the following reasons. First, if we scale out the cluster late, more VMs should be added to guarantee the deadline. However, as shown in our previous experiment, more VMs lead to higher costs. Second, VMs cannot be set up instantly. Thus, if we scale out the cluster late, the overhead of adding a newly set-up VM into the cluster will take a period of time (e.g., 300 seconds in our experiment), leading to insufficient preparation time before VMs start to work. Additionally, the scaling trigger point also should be robust against the predicted completion time spike caused by noise in data.

To this end, we introduce two thresholds, $r_{threshold}$ and $t_{threshold}$, to identify an appropriate scaling trigger point. The conditions of triggering a scaling operation are defined as: if the ratio of the predicted completion time to the deadline is later than a value (i.e., $r_{threshold}$) and this phenomenon lasts longer than a period (i.e., $t_{threshold}$), then we scale out the cluster. In practice, the settings of $r_{threshold}$ and $t_{threshold}$ need fine-tuning. In the following evaluation section, we will evaluate how $r_{threshold}$ and $t_{threshold}$ may affect the performance and how to fine-tune these two thresholds.

Scale Controller has to determine the minimum number of VMs to be scaled out when a scaling operation becomes necessary. The main challenge for this determination is that additional VMs cannot be effective to the cluster instantly.

Algorithm 1 Minimum number of scaling VMs.

Require: V_1=$[w, n, R, H, U, G]$; V_2=$[M_P, R_P, M_S, R_S, T_P, T_I]$; V_3=$[H_C, H_L, H_M, W_C, W_L, W_M]$;
$p(\cdot)$: prediction model of Scale Controller;
$t_{threshold}$; $t_{overhead}$; $t_{deadline}$;
Ensure: Δn;
Update $V_1^* = [w, n^*, R, H, U, G]$, where $n^* = n + 1$;
Calculate the locally weighted value of M_S, R_S and T_I during last $t_{threshold}$ as $\overline{M_S}$, $\overline{R_S}$ and $\overline{T_I}$;
Update V_2^*=$[M_P^*, R_P^*, \overline{M_S}, \overline{R_S}, T_P^*, \overline{T_I}]$, where $M_P^* = M_P + \overline{M_S} \cdot t_{overhead}$, $R_P^* = R_P + \overline{R_S} \cdot t_{overhead}$ and $T_P^* = T_P + \frac{1}{\overline{T_I}} \cdot t_{overhead}$;
Update $V_3^* = V_3$;
while $p(V_1^*, V_2^*, V_3^*) > t_{deadline}$ **do**
$n^* = n^* + 1$;
$\Delta n = n^* - n$;
return Δn;

It takes about $t_{overhead}$ to actually launch VMs and make the computing resources available to the computing job. Before newly added VMs become effective, the running status might have changed, which makes the newly added resources either more than necessary or insufficient.

To address this issue, we propose an algorithm to obtain the minimum number of VMs for scaling, which including two steps as shown in Algorithm 1. The first step is to get a proper estimation about running status, V_1^*, V_2^*, and V_3^*, before the newly added VMs are effective. Since the provisioning time, $t_{overhead}$, is rather shorter compared to the overall running time, we assume that the running speed is steady during $t_{overhead}$ and the resource utilization is roughly the same (i.e., $V_3^* = V_3$). Other features in V_1 need not update except the number of VMs will change. Thus, we only need to update V_2^*, which can be calculated by the sum of the current V_2 and the processed percentage during $t_{overhead}$. However, in order to calculate the processed percentage, $\overline{M_S}$, $\overline{R_S}$ and $\overline{T_I}$ should be appropriately estimated. A good estimation should not be an outlier. It should well present the running speed after the performance degradation. Since the later time points during $t_{threshold}$ are closer to the trigger point, their value is more accurate to estimate the running speed in $t_{overhead}$. Finally, we select the Locally Weighted Value of M_S in the last $t_{threshold}$ as $\overline{M_S}$. For all the sample points during the last $t_{threshold}$, their weight can be calculated as follow:

$$W_t = exp(-\frac{(t_{threshold} - t)^2}{2c}), \tag{7}$$

where c is a parameter added to control the importance of samples on a different timeline. In Eq. 7, W_t has a larger value if it is closer to the trigger point. Then, we can calculate $\overline{M_S}$ as follows:

$$\overline{M_S} = M_S(t) \cdot W_t / \sum W_t, \tag{8}$$

where $\overline{R_S}$ and $\overline{T_I}$ can be calculated in a similar way by just replaces $\overline{M_S}$ by $\overline{R_S}$ or $\overline{T_I}$ respectively. The second step is to iteratively increase n^* in V_1^* until Scale Controller's predicted completion time is within the deadline. As a result, the difference between n^* and n is the minimum necessary number of VMs.

4 Performance Evaluation

In this section, we first introduce the data set and experiment setup. We then investigate the performance of Scale Estimator and Scale Controller with extensive experiments. Finally, through real experiments compared to the state-of-the-art, we demonstrate the superiority of MarVeLScaler.

4.1 Data Set and Experiment Setup

Data Set. The raw data used for training our model, including three kinds of representative MapReduce workloads (i.e., WordCount, TeraSort, and PageRank). We run each workload for 100 times and finally collect 300 groups of data. Each group consists of 3 kinds of raw data: 1) Records of computing resources and running settings; 2) Logs collected with Log4j [31], an original log tool of MapReduce framework; 3) Resource utilization data collected with Ganglia Monitoring System [32].

With straightforward feature engineering (i.e., normalization and discretization), we transform the records of computing resources as input feature vectors and the completion time as labels of Scale Estimator. 80% of the data are randomly selected as training set, while the rest as the evaluation set. Finally, we get 100 groups of data for each workload to construct the regression model.

The feature engineering of Scale Controller involves some important preprocesses. Since the logs and resource utilization data are consecutive data, we first leverage a sample window of 5 seconds to divide these data into several small fragments. Then, we calculate the average values of the features in Table 5 and Table 6, and obtain approximately 30,000 groups of data. Thus, the data set for Scale Controller is much larger and contains more useful information. Next, through normalization and proper padding, these data are suitable inputs of LSTM structures. As for resources features and running setting features, a similar feature process approach in Scale Estimator is applied. Finally, 3 views of training data are obtained. We randomly choose data from 80% jobs as the training set and the rest as the evaluation set, which is a common setting in time-series data mining [33] [34].

Experiment Setup. We implement MarVeLScaler, a prototype auto-scaling system in Alibaba Cloud with at most 40 VMs, where each VM is equipped with 4 Intel Xeon E5-2682 v4 2.5GHz CPU. The version of Hadoop on the VM is 2.7.2. This online prototype system is written in Python 3.5, consisting of two threads, the *Monitor* thread for collecting the information of the whole cluster and the *Predictor* thread for online completion time predictions. Since the trace of our experiment is generated, we need to set up proper deadlines for the executed jobs in our training data. We set the predicted completion time of Scale Estimator as the deadline due to the following two reasons. First, Scale Estimator works in an ideal situation where no performance degradation occurs. If we set the result of Scale Estimator as a deadline, jobs that experience performance degradation will exceed the deadline. Second, Scale Estimator tries to

TABLE 7: The statistics of our regression model.

Symbol	Value	Symbol	Value
R^2	0.9772	Chi-Square	23.5856
F-Statistic	554.3763	RMSE	19.4366
MRE	98.4%		

TABLE 8: The results of Scale Estimator for resource recommendation.

No. of VMs	Mem (GB)	Storage	CPU	Time (s)	Cost ($)
40	16	HDD	Exclusive	669.2	2.37
40	16	SSD	Exclusive	663.6	2.72
40	8	HDD	Exclusive	692.4	2.29
40	8	SSD	Exclusive	686.8	2.63
39	16	HDD	Exclusive	672.9	2.31
39	16	SSD	Exclusive	678.5	2.66
39	8	SSD	Exclusive	696.1	2.23
38	16	HDD	Exclusive	688.4	2.25
38	16	SSD	Exclusive	682.8	2.60
37	16	HDD	Exclusive	699.2	2.20
37	16	SSD	Exclusive	693.6	2.53

provide the initial most cost-saving option (as shown in Table 8) for tenants to set up the cloud cluster. And we choose the resource allocation with a completion time of 699.2s, while the deadline is 700s. Overall, it is reasonable to use the result of Scale Estimator as the deadline.

4.2 Evaluation of Scale Estimator

4.2.1 Statistics of the Regression Model

To examine the effectiveness of the regression model, we first obtain several commonly used statistics [35] in Table 7. The coefficient of determination, R^2, is the proportion of the variance in the dependent variables, which can be used to test the fitness of the regression model. R^2 takes values from 0 (bad fitness) to 1 (good fitness). In the non-linear fitting, the value of Chi-Square is generated under the best values of all coefficients (less is better). Then, the F-statistic is used when comparing statistic models that have been fitted to a data set. It indicates the regression model that best fits the population from which training data were sampled. The Root-Mean-Square-Error (RMSE) is the prediction errors between the observed value and the ground truth. Finally, MRE denotes the mean relative error of the regression model.

From Table 7, we can see that R^2 (0.9772), RMSE (19.4366) and MRE (98.4%) of our regression model are rather good. To be specific, 0.9772 of R^2 indicates good fitness because its value is larger than 0.95. From the perspective of RMSE and MRE, the predictions of our model are extremely close to the ground-truth value. While Chi-Square (23.5856) and F-statistic (554.3763) are model-dependent, which means the interpretation of these values varies according to different models. In our case, the values of these two statistics are the best we can get in our experiment. Overall, the values in Table 7 validate the efficacy of our proposed regression model.

4.2.2 Effect of Scale Estimator

Based on the above regression model, we implement Scale Estimator which aims to generate a detailed resource purchase list for tenants to set up the cloud cluster for performing MapReduce jobs. To demonstrate the effectiveness, we conduct a case study by submitting a WordCount job with a workload of 400 GB and a deadline of 700 seconds to the Scale Estimator.

To be specific, we first search all the resource allocation combinations with at most using 40 VMs where makes a job finished by the deadline will be recorded in a list. The results are shown in Table 8, we can see that the row with the underline is the resource recommendation of our Scale Estimator, which is the most cost-saving choices (i.e., including 37 VMs and each equipped with 16 GB of memory, HDD storage and exclusive CPU) for the tenants to set up the cloud cluster in terms of current submitted workload and deadline. The reason behind this is that our regression model can well capture the initial relationship among the workload, the computing resources and the completion time of a job. Overall, the case study successfully validate the efficacy of our proposed Scale Estimator.

TABLE 9: Prediction error (RMSE) of different views based neural network model on WordCount workload.

$Diff$	0-10%	10-20%	20-30%	>30%	>0%
V_1	1443.08	2995.16	5303.61	6273.25	1985.29
V_2	3360.32	3482.78	3880.8	4015.4	3657.74
V_3	Failed				
Multi-view	1337.52	1960.8	3470.85	4158.55	1506.17

4.3 Evaluation of Scale Controller

4.3.1 Neural Network Model Accuracy

To demonstrate the promotion of prediction accuracy by using multi-view instead of a single view. We are to train 4 neural network models with different views of input features (i.e., V_1, V_2, V_3 and Multi-view), respectively. Meantime, we measure the differences (i.e., the degree of VM's performance degradation), denoted as $Diff$, which indicates the difference between real completion time (R_t) and deadline (D_t) of jobs that exceed the deadline. Formally,

$$Diff = \frac{|R_t - D_t|}{D_t}.$$

According to the *Diff* metric, we divide the evaluation data set into 5 groups (i.e., 0-10%, 10-20%, 20-30%, >30%, >0%). Next, we test the prediction error (RMSE) of each neural network on each group and give a detailed analysis.

As we can see in Table 9, the prediction error of V_1 significantly increases with *Diff* (from 1443.08 to 6273.25), which indicates bad prediction performance when the performance of VMs degrades. This is because the features in V_1 are static, and they cannot capture the performance variance of VMs. The prediction error of V_2 is rather steady, but the overall error (3657.74) of this view is rather severe. The reason is that the features in V_2 will violently fluctuate before they get a stable state, which will introduce significant errors. Moreover, the model based on V_3 fails in learning, because utilization cannot predict the completion time alone. Our multi-view model always outperforms the models based on V_1 and V_2 in terms of the prediction error under different *Diff* conditions. For example, considering the overall average prediction errors, our multi-view model is 1506.17, which is an improvement of 0.32 and 2.43 times

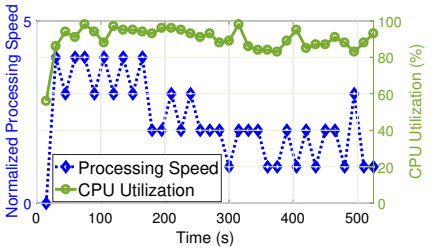


Fig. 7: The running speed and CPU utilization as a function of time.

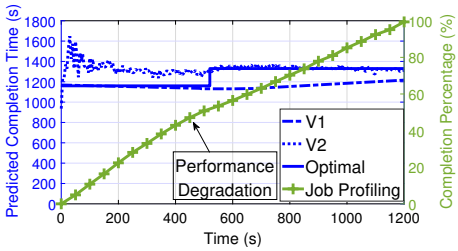


Fig. 8: The prediction results of using single-view features.

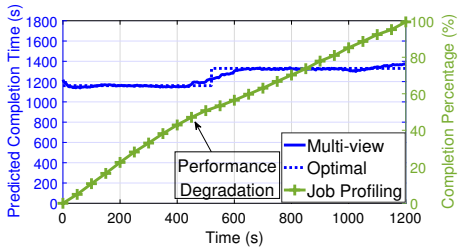


Fig. 9: The prediction results of using multi-view features.

over the V_1 and V_2 based neural network, respectively. That is because, our multi-view model takes the advantage to comprehensively capture the running status of a MapReduce job by extracting key information from three different groups of features (i.e., computing resources, resource utilization, and processing speed).

4.3.2 Effect of the Multi-view Model

For further study of the effectiveness of our multi-view deep learning model, we conduct a case study by using the neural networks above to predict the completion time of a MapReduce job (e.g., 20 VMs for 400 GB workload) with obvious performance degradation. We first profile a job with obvious performance degradation at 200s, which is shown in Fig. 7. We see that the current resource utilization based auto-scaling rules (e.g., to scale when CPU utilization is higher than 80%) cannot work well on MapReduce jobs. That is because there exist no obvious changes in CPU utilization when the performance degrades.

Next, we give a formal definition of *Optimal Prediction*, which means capturing performance degradation immediately and adjust its result according to the running speed. As we can see in Fig. 8. The variation in the running speed line indicates when the performance degradation occurs (e.g., 500s in this case). During the first 500 seconds, predictions from V_1 are accurate and almost fit the *Optimal Prediction*, while the predictions of V_2 varies significantly and tend to make unnecessary scaling decisions. After the occurrence of performance degradation, predictions from V_1 cannot follow the *Optimal Prediction*, leading to the absence of timely scaling operations, while predictions of V_2 gradually get stable and show stable predicted completion time according to the real-time running status. However, in Fig. 9, we can see that our multi-view model performs well all the time and shows the ability to make timely and accurately scaling decisions. That is because our multi-view model can capture the running status to overcome the deadline violation problem caused by VM performance degradation.

4.3.3 Scaling Timing and Amount

As discussed in the previous section, the settings of $r_{threshold}$ and $t_{threshold}$ are very crucial to the effectiveness of our Scale Controller. In the following, we are to evaluate how $r_{threshold}$ and $t_{threshold}$ affect the performance and how to fine-tune these two thresholds.

First, we give a formal definition of P, which indicates the ratio of the number of jobs with correct scaling operations to the number of total jobs. Formally,

$$P = \frac{P_1 + P_2}{N},$$

where N denotes the total number of jobs; P_1 and P_2 denote the number of jobs with performance degradation while Scale Controller makes scaling decisions, and the number of jobs without performance degradation while Scale Controller does not make scaling decisions, respectively.

Next, we are to test P with different $r_{threshold}$ and $t_{threshold}$. As we can see in Fig. 10, the combination of $r_{threshold}$ and $t_{threshold}$ greatly affects the scaling accuracy. We first focus on $r_{threshold}$. For most lines, the highest P are obtained when $r_{threshold}$ is 0.85. Then, P drops gradually with the increment of $r_{threshold}$, and achieves its lowest value when $r_{threshold}$ is 1. After that, P increases to about 90%, which indicates a satisfactory prediction result. The lowest P are obtained when $r_{threshold}$ is 1, which means it is improper to scale out the cluster once the predicted completion time is larger than the deadline. The reason is that this setting cannot deal with the prediction errors well. Through our observation, a considerable part of jobs can be finished around the deadline, while their predicted completion time fluctuates beyond and below the deadline. When the completion time of a job exceeds the deadline slightly, this fluctuation will make the system extremely difficult to determine a proper scaling operation. However, when $r_{threshold}$ is 0.85, the determination of scaling operation would be easier. The aforementioned jobs will operate well under this setting. However, it also leads to excessive cost since it tends to make unnecessary scaling decisions. When $r_{threshold}$ is 1.15, the scaling operation would be even harder to decide. The advantage of this setting is that it prevents some unnecessary scaling operations, which saves much cost. As for $t_{threshold}$, a higher $t_{threshold}$ would be better since it leads to enough time to obtain running speed parameters (will show later). Considering P and cost, we set $r_{threshold}$ at 1.1 and $t_{threshold}$ at 60s. In this situation, we get an accurate enough P and rather low cost. Besides, the choice of $t_{threshold}$ will decide a timely scaling operation and the reasonable amount of resources.

Finally, after determining the scaling time point, Scale Controller needs to decide the proper number of scaling VMs, while keeping the payment at a minimum. Note that the overhead of adding additional VMs to the cloud cluster is rather a long period of time (e.g., 300s) compared to the completion time of most jobs. In order to obtain a more

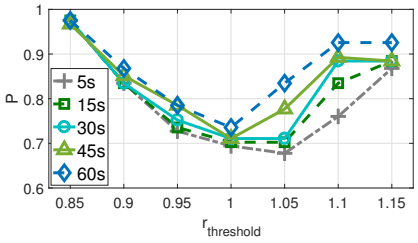


Fig. 10: P with different $r_{threshold}$ and $t_{threshold}$.

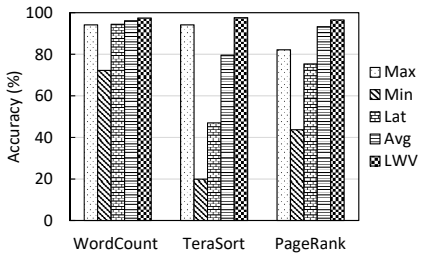


Fig. 11: Prediction accuracy of different $\overline{M_S}$, $\overline{R_S}$ and $\overline{T_I}$.

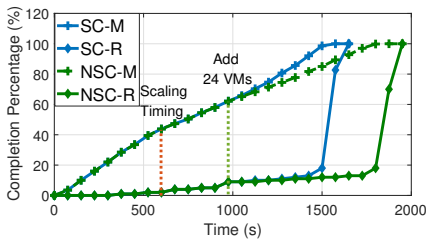


Fig. 12: The scaling effect of Scale Controller.

TABLE 10: The average cost, average tardiness and the number of missed deadline jobs of MarVeLScaler and baselines.

Workload Type	WordCount				TeraSort				PageRank			
Method	BS1	BS2	BS3	Our	BS1	BS2	BS3	Our	BS1	BS2	BS3	Our
$AvgCost$ (VM*s)	10501	14134	12531	11199	11239	15457	13573	11649	11206	16677	12600	11317
$AvgTard$ (s)	251.8	13.3	125.1	20.0	81.2	11.0	29.3	11.3	127.8	11.5	38.6	15.7
No. of Missed Deadline Jobs	6	1	4	1	4	1	2	1	5	1	2	1

precise number of scaling VMs, we take this overhead into account. As explained in Section 3.3.3, the most critical thing is to predict V_2^*. Hence, we test V_2^* under different statistic parameters of M_S, R_S and T_I in terms of prediction accuracy, which consists of the following baselines: 1) the maximum value (Max); 2) the minimum value (Min); 3) the latest value (Lat); 4) the average value (Avg); 5) the locally weighted value (LWV). In Fig. 11, we see that except for the minimum value, other statistical methods all achieve relatively good estimations. Notably, the locally weighted value achieves the highest accuracy among them. That is because it correctly captures the importance of the latter value and dampens the influence of outliers. Thus, we choose the locally weighted value of M_S, R_S and T_I as our $\overline{M_S}$, $\overline{R_S}$ and $\overline{T_I}$ to calculate the scaling amount in Algorithm 1.

4.3.4 *Effect of Scale Controller*

Based on the above multi-view deep learning model, we implement Scale Controller which aims to adjust the scale of the cluster according to its real-time running status to guarantee the MapReduce job finished on time. To demonstrate the effectiveness, we conduct a case study by submitting a WordCount job with a workload of 400 GB and a deadline of 1700 seconds to the Scale Controller.

To be specific, the initial size of the cloud cluster is 16 VMs according to the recommendation of Scale Estimator. We employ the following baselines, namely, NSC-M and NSC-R, which represent the running procedure of Map and Reduce stage without scaling, respectively. Additionally, SC-M and SC-R are the actual running procedure under the control of Scale Controller. As we can see in Fig. 12, the scaling timing appears at 675s, Scale Controller finds that another 24 VMs should be added to the cluster to meet the deadline. The cluster obtains these VMs at 975s. After that, the running speed (in terms of the increasing speed of competition percentage) of the job boosts a lot, and finally meets the deadline at 1650s. In contrast, the baseline exceeds the deadline with a completion time of 1950 seconds. The reason behind is that our multi-view deep learning model can timely capture the performance degradation of VMs and make an auto-scaling decision whenever necessary. Overall, the case study successfully validates the efficacy of our proposed Scale Controller.

4.4 Evaluation of MarVeLScaler

The basic setup for the experiment is introduced in previous sections. In the following, we are to evaluate the performance of MarVeLScaler with 3 different workloads (i.e., WordCount, TeraSort, and PageRank), where each workload contains 200 GB data and will be tested over 10 times.

Comparison Baselines. We select the following state-of-the-art models as baselines. It is worth noting that the first two are some of the static models, and the last two are the dynamic models.

- **Baseline 1 (BS1)** is generated from [9] which reserves 10% of resources as extra resources to avoid missing the deadline.
- **Baseline 2 (BS2)** uses the same static model as BS1 but with reserving 30% of resources as extra resources.
- **Baseline 3 (BS3)** conducts auto-scaling decisions based on a neural network [11], while only uses job-related features to make a prediction.
- **MarVeLScaler (Our)** leverages a multi-view deep learning model to conduct real-time scaling decisions making for guaranteeing MapReduce jobs finished on time in public clouds.

Evaluation Metrics. For a resource allocation prediction method, the ability to provide credible cost saving is crucial to every tenant. Cost is always the first concern, and we measure the Average Cost (AvgCost), which measures the average cost of renting cloud resources to complete a job before the deadline. Formally,

$$AvgCost = \frac{1}{|J|}\sum_{j=1}^{J}\sum_{i=1}^{n} P_i \cdot t_i,$$

where J denotes the job set, i.e., $J = \{j\}$, which consists of a series of jobs; n denotes the number of VMs requested from cloud service providers; P_i and t_i denote the price and processing time of the i_{th} VM, respectively.

It is also significant that our job should be finished before the deadline, and we also measure the Average Tardiness (AvgTard), which indicates the difference between the real completion time and the deadline. Formally,

$$AvgTard = \frac{1}{|J|}\sum_{j=1}^{J}(R_j - D_j),$$

where $R_j - D_j = \begin{cases} 0, & \text{if } R_j \leq D_j \\ R_j - D_j, & \text{otherwise} \end{cases}$; R_j and D_j denote the real completion time and deadline of the j_{th} job, respectively.

Experiments Results. Table 10 shows the Average Cost, Average Tardiness and number of missed deadline jobs of performing the same MapReduce job multiple times under three baselines and MarVeLScaler. First, we can see that MarVeLScaler always outperforms BS2 and BS3 in terms of average cost under different kinds of MapReduce jobs. As for the BS1, it gets less costs because of sacrificing not being able to meet deadlines (will be shown later). Second, from the average tardiness point of view, we can see that MarVeLScaler is comparable with BS2, which outperforms BS1 and BS3 by almost 11 and 5 times when performing WordCount job. Finally, we see that MarVeLScaler exceeds the deadline far less than other state-of-the-art methods. Similar conclusions can be drawn for other workloads. Overall, MarVeLScaler saves more money while keeping a better ability to finish jobs by the deadline. That is because MarVeLScaler applies multi-view deep learning techniques to solve the inaccurate prediction issue caused by performance degradation of VMs and conduct the auto-scaling decisions whenever necessary. As for MarVeLScaler still misses some deadlines, the reason presumably is that it only becomes clear that the deadline will be missed when it's already too late to act. It would be an interesting future work to further improve our MarVeLScaler's performance to ensure never miss the deadlines.

5 Related Work

Recently, there is a trend of using knowledge-based models to better estimate the completion time of a specific workload. Considering the different objectives, we classify these models into the following two categories, namely static models and dynamic models.

Static Models are mainly used for resource reservation and job scheduling. Jalaparti et al. [36] use the knowledge from historical executed jobs to plan and coordinate the placement of data and tasks. Lim et al. [37] focus on processing an open stream of MapReduce jobs with Service-Level Agreements (SLAs). To schedule the open stream jobs, the authors devise MRCP-RM, which is a constraint programming based resource management scheme. Additionally, a data locality-aware resource management algorithm is designed to ensure that the whole system can satisfy SLAs. Jockey [38], ARIA [39], and CRESP [9] also fall into this category. These works explore different methods of acquiring the quantitative relationship between job-related parameters and deadline. However, these static models cannot be directly applied for tenants to set up the cloud cluster. Our Scale Estimator leverages machine learning (i.e., regression model) techniques to capture the quantitative relationship among the workload, the computing resources and the completion time of a job, which avoid the detailed analyses of MapReduce processing procedure. Notably, Scale Estimator is built in a more resource fine-grained way, so it can be directly applied to generate a detailed resource purchase list for tenants to set up the cloud cluster for performing MapReduce jobs.

Dynamic Models are suitable for online resource adjustment. MLscale [40] aims to satisfy SLAs of interactive applications (e.g., web server), by using a neural network to model the relationship between monitored metrics and the performance metrics (e.g., response time), and a regression-based model to predict performance after scaling. AGILE [14] introduces another model (i.e., polynomial curve fitting) to conduct resource demand prediction and set up VMs proactively to avoid performance degradation. Reinforcement learning based models like [41] can transform the problem of packing tasks with multiple resource demands into an optimization problem, demonstrating good adaption, quick convergence, and learning of smart strategies. Xu et al. [16] propose a theoretic model to provide hard deadline guarantees for cloud-based MapReduce. However, these dynamic models ignore the performance degradation of VMs, which may lead to wrong scaling decisions. This work is the first attempt on applying multi-view deep learning techniques to solve the inaccurate prediction issue caused by the performance degradation of VMs. Specifically, we applies a Deep Canonically Correlated AutoEncoder (DCCAE) [19] structure to refine a high-level representation from the raw multi-view features (i.e., offline and online features) to simultaneously learn different views of features and maximize the correlations of features across views, which sheds some new light on real-time scaling decision making for guaranteeing MapReduce jobs finished on time in public clouds.

6 Discussion

Naturally, there is room for further work and possible improvements. We discuss a few points here.

Dynamic Resource Provisioning Implementation in Hadoop. Hadoop is a distributed computing framework, which supports online dynamic resource provisioning. It means that the original cluster needs not to suspend the computing job, while additional VMs are added to the cluster. Specifically, the newly added VM should first configure its host name, running environment and Hadoop operating user, etc. Then, the VMs in the original VMs should include the host name and IP address of the newly added VMs. Finally, when the Hadoop service on each new VM is launched, these VMs can be used to accelerate the computing job. In our system, we implement the dynamic resource provisioning function with the APIs from Alibaba E-MapReduce. More details can be found in [42]

Auto-scaling with Homogeneous VMs Scenarios. In this study, we focus on auto-scaling merely with homo-

geneous VMs scenarios, which is a typical practice for IaaS Cloud tenants. First, the vast majority of auto-scaling researches, such as [43] [44] [45], are assumed that VMs are homogeneous. Second, when tenants execute their computation jobs on multiple VMs with different configurations in practice, some problems will be raised. Take MapReduce as an example, it has at least two phases. If we run Map phase on those VMs with the same configuration, and then feed the output to the Reduce phase while running on the VMs with different configurations, those outputs may not be compatible to those VMs in the Reduce phase. Moreover, fatal errors may occur during execution. To avoid these potential problems, in this study, we design our regression model merely for homogeneous VMs scenarios. It would be an interesting future work to extend our approach to those heterogeneous VMs scenarios.

Linear Relationship Assumption in Regression Model. In this study, our regression model is designed to generate a detailed initial resource purchase list for tenants to set up the cloud cluster for performing MapReduce jobs. Typically, the list consists of the features such as H, U, R and G in Table 3, which usually represent the computing ability of a VM. We assume that these features have a linear relationship due to the following reasons. Firstly, when we are constructing a regression model, there must be given an exact form, which is unlike neural networks that can be trained to automatically eliminate irrelevant features according to the data. Secondly, we have tried to assume other forms (e.g., non-linear), while those are kind of less effective than the linear assumption in terms of prediction accuracy. Finally, through the Pearson correlation coefficients (PCC) test, these features have much less impact on the predicted completion time, which means our assumption will not significantly deviate the predicted value from the ground truth. Exploring other models (e.g., neural network) would be an interesting future work for generating the initial resources purchase list for tenants.

7 Conclusions

In this paper, we present *MarVeLScaler*, a prediction based prototype auto-scaling system including two useful modules, namely, *Scale Estimator* and *Scale Controller*. Scale Estimator leverages a regression model to preliminarily estimate the required cluster size for a MapReduce job with given a concrete workload and deadline. During the runtime, Scale Controller leverages a multi-view deep learning model to adjust the scale of the cluster according to its real-time running status to guarantee the job finished on time. We evaluate the performance of MarVeLScaler based on Hadoop in Alibaba Cloud. The experiment results show that MarVeLScaler can provide 98.4% accuracy of prediction in determining initial cluster size, and save 30.8% of expense while still guaranteeing similar performance compared with the state-of-the-art methods. We believe that our MarVeLScaler offers an effective and practical mechanism for reducing the required expenses associated with performing MapReduce jobs in public clouds.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation Comp. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[2] A. C. Zhou and B. He, "Transformation-based monetary costoptimizations for workflows in the cloud," *IEEE Trans. Cloud Computing*, vol. 2, no. 1, pp. 85–98, 2014.

[3] J. Schad, J. Dittrich, and J. Quiané-Ruiz, "Runtime measurements in the cloud: Observing, analyzing, and reducing variance," *PVLDB*, vol. 3, no. 1, pp. 460–471, 2010.

[4] E. Casalicchio and L. Silvestri, "Mechanisms for SLA provisioning in cloud-based service providers," *Computer Networks*, vol. 57, no. 3, pp. 795–810, 2013.

[5] H. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th International Conference on Autonomic Computing, ICAC 2010, Washington, DC, USA, June 7-11, 2010*, 2010, pp. 1–10.

[6] J. Zhao, C. Xue, X. Tao, S. Zhang, and J. Tao, "Using adaptive resource allocation to implement an elastic mapreduce framework," *Softw., Pract. Exper.*, vol. 47, no. 3, pp. 349–360, 2017.

[7] R. Han, L. Guo, M. M. Ghanem, and Y. Guo, "Lightweight resource scaling for cloud applications," in *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid)*. IEEE Computer Society, 2012, pp. 644–651.

[8] M. M. Murthy, H. Sanjay, and J. Anand, "Threshold based auto scaling of virtual machines in cloud environment," in *IFIP International Conference on Network and Parallel Computing*. Springer, 2014, pp. 247–256.

[9] K. Chen, J. Powers, S. Guo, and F. Tian, "CRESP: towards optimal resource provisioning for mapreduce computing in public clouds," *IEEE TPDS*, vol. 25, no. 6, pp. 1403–1412, 2014.

[10] L. Yazdanov, M. Gorbunov, and C. Fetzer, "Ehadoop: Network I/O aware scheduler for elastic mapreduce cluster," in *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*, 2015, pp. 821–828.

[11] A. Gandhi, S. Thota, P. Dube, A. Kochut, and L. Zhang, "Autoscaling for hadoop clusters," in *IEEE International Conference on Cloud Engineering, IC2E , Berlin, Germany, April 4-8*, 2016, pp. 109–118.

[12] N. Chalvantzis, I. Konstantinou, and N. Koziris, "BBQ: elastic mapreduce over cloud platforms," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, Madrid, Spain, May 14-17, 2017*, 2017, pp. 766–771.

[13] Hadoop, "SlowStart". http://hadoop.apache.org/docs/r2.4.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/mapred-default.xml, 2019.

[14] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes, "AGILE: elastic distributed resource scaling for infrastructure-as-a-service," in *10th International Conference on Autonomic Computing, ICAC'13, San Jose, CA, USA, June 26-28, 2013*, 2013, pp. 69–82.

[15] V. Aggarwal, M. Xu, T. Lan, and S. Subramaniam, "On the optimality of scheduling dependent mapreduce tasks on heterogeneous machines," *CoRR*, vol. abs/1711.09964, 2017.

[16] X. Xu, M. Tang, and Y. Tian, "Theoretical results of qos-guaranteed resource scaling for cloud-based mapreduce," *IEEE Trans. Cloud Computing*, vol. 6, no. 3, pp. 879–889, 2018.

[17] D. Cheng, X. Zhou, Y. Xu, L. Liu, and C. Jiang, "Deadline-aware mapreduce job scheduling with dynamic resource availability," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 814–826, 2019.

[18] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," in *ACM Symposium on Cloud Computing in conjunction with SOSP 2011, SOCC '11, Cascais, Portugal, October 26-28, 2011*, 2011, p. 18.

[19] W. Wang, R. Arora, K. Livescu, and J. A. Bilmes, "On deep multi-view representation learning," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 1083–1092.

[20] A. M. Elkahky, Y. Song, and X. He, "A multi-view deep learning approach for cross domain user modeling in recommendation systems," in *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 2015, pp. 278–288.

[21] S. Li, Y. Li, and Y. Fu, "Multi-view time series classification: A discriminative bilinear projection approach," in *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, 2016, pp. 989–998.

[22] I. V. Tetko, D. J. Livingstone, and A. I. Luik, "Neural network studies. 1. comparison of overfitting and overtraining," *Journal of*

chemical information and computer sciences, vol. 35, no. 5, pp. 826–833, 1995.
[23] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of biomedical informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.
[24] J. Rodgers and A. Nicewander, "Thirteen ways to look at the correlation coefficient," *American Statistician - AMER STATIST*, vol. 42, pp. 59–66, 02 1988.
[25] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.
[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[27] G. Andrew, R. Arora, J. A. Bilmes, and K. Livescu, "Deep canonical correlation analysis," in *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013, pp. 1247–1255.
[28] T. D. Bie and B. D. Moor, "On the regularization of canonical correlation analysis," in *Proceedings of the International Conference on Independent Component Analysis and Blind Source Separation*, 2003.
[29] W. Wang, R. Arora, K. Livescu, and J. A. Bilmes, "Unsupervised learning of acoustic features via deep canonical correlation analysis," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 4590–4594.
[30] N. Mallinar and C. Rosset, "Deep canonically correlated lstms," *CoRR*, vol. abs/1801.05407, 2018.
[31] "Apache log4j 2," https://logging.apache.org/log4j/2.x/.
[32] "Ganglia monitoring system," http://ganglia.info/.
[33] Y. Tong, Y. Chen, Z. Zhou, L. Chen, J. Wang, Q. Yang, J. Ye, and W. Lv, "The simpler the better: a unified approach to predicting original taxi demands based on large-scale online platforms," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2017, pp. 1653–1662.
[34] Z. Zheng, Q. Chen, C. Fan, N. Guan, A. Vishwanath, D. Wang, and F. Liu, "Data driven chiller sequencing for reducing hvac electricity consumption in commercial buildings," in *Proceedings of the Ninth International Conference on Future Energy Systems*. ACM, 2018, pp. 236–248.
[35] A. C. Cameron and F. A. Windmeijer, "An r-squared measure of goodness of fit for some common nonlinear regression models," *Journal of Econometrics*, vol. 77, no. 2, pp. 329 – 342, 1997.
[36] V. Jalaparti, P. Bodík, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proceedings of the ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 407–420.
[37] N. Lim, S. Majumdar, and P. Ashwood-Smith, "MRCP-RM: A technique for resource allocation and scheduling of mapreduce jobs with deadlines," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 5, pp. 1375–1389, 2017.
[38] A. D. Ferguson, P. Bodík, S. Kandula, E. Boutin, and R. Fonseca, "Jockey: guaranteed job latency in data parallel clusters," in *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference, Bern, Switzerland, April 10-13*, 2012, pp. 99–112.
[39] A. Verma, L. Cherkasova, and R. H. Campbell, "ARIA: automatic resource inference and allocation for mapreduce environments," in *Proceedings of the 8th International Conference on Autonomic Computing, Karlsruhe, Germany, June 14-18, 2011*, 2011, pp. 235–244.
[40] M. Wajahat, A. Gandhi, A. Karve, and A. Kochut, "Using machine learning for black-box autoscaling," in *Seventh International Green and Sustainable Computing Conference, IGSC*, 2016, pp. 1–8.
[41] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks, HotNets 2016, Atlanta, GA, USA, November 9-10, 2016*, 2016, pp. 50–56.
[42] "Alibaba cloud, e-mapreduce," https://www.alibabacloud.com/help?spm=a3c0i.8276058.1097638.dnavdocumentation0.5f0f2ca7hzNky2.
[43] Y. Seung, T. Lam, L. E. Li, and T. Woo, "Cloudflex: Seamless scaling of enterprise applications into the cloud," in *2011 Proceedings IEEE INFOCOM*. IEEE, 2011, pp. 211–215.
[44] P. Tang, F. Li, W. Zhou, W. Hu, and L. Yang, "Efficient auto-scaling approach in the telco cloud using self-learning algorithm," in *IEEE Global Communications Conference*. IEEE, 2015, pp. 1–6.
[45] C. Qu, R. N. Calheiros, and R. Buyya, "Auto-scaling web applications in clouds: A taxonomy and survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 4, p. 73, 2018.

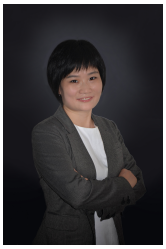
Yi Li received his B.Eng. degree in the College of Electronic Information Engineering, Wuhan University of Science and Technology, China. He is currently a M.Eng. student in the School of Computer Science and Technology, Huazhong University of Science and Technology, China. His research interests include cloud computing and machine learning.

Fangming Liu received the B.Eng. degree from the Tsinghua University, Beijing, and the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a Full Professor with the Huazhong University of Science and Technology, Wuwhan, China. His research interests include cloud computing and edge computing, datacenter and green computing, SDN/NFV/5G and applied ML/AI. He received the National Natural Science Fund (NSFC) for Excellent Young Scholars, and the National Program Special Support for Top-Notch Young Professionals. He is a recipient of the Best Paper Award of IEEE/ACM IWQoS 2019, ACM e-Energy 2018 and IEEE GLOBECOM 2011, as well as the First Class Prize of Natural Science of Ministry of Education in China.

Qiong Chen received his B.Eng. degree in School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. He is currently a M.Eng. student in School of Computer Science and Technology, Huazhong University of Science and Technology. His research interests include applied machine learning and edge computing. He received the Best Paper Award of ACM International Conference on Future Energy Systems (ACM e-Energy) in 2018.

Yibing Sheng is studying as B.Com. in University of Toronto, double specialists in Computer Science under the Faculty of Arts and Science and Finance and Economics under Rotman Commerce. His interests include machine learning and networking.

Miao Zhao is an assistant professor at Department of Computing, The Hong Kong Polytechnic University. She received her Ph.D. from Department of Electrical and Computer Engineering, State University of New York, Stony Brook, NY in 2010. She received her bachelor and master degrees from Department of Electronics and Information Engineering of Huazhong University of Science and Technology, Wuhan, China. Her current research interests focus on data mining, representation learning, recommendation, multimedia analytics and networking.



Jianping Wang is a professor of the computer science department at City University of Hong Kong, Hong Kong. She received the B.E. degree and MSc degrees in computer science from Nankai University, Tian Jin, China, in 1996 and 1999, respectively and the Ph.D. degree in computer science from University of Texas at Dallas, USA, in 2003. Her research interests include cloud computing, service oriented networking, and data center networks.