Cluster Computing (2018) 21:1607–1622
https://doi.org/10.1007/s10586-018-2265-1

Multi-prediction based scheduling for hybrid workloads in the cloud data center

Haiou Jiang[1] · Haihong E[1] · Meina Song[1]

Received: 2 May 2016 / Revised: 9 October 2017 / Accepted: 19 February 2018 / Published online: 4 June 2018

Abstract
Cloud computing can leverage over-provisioned resources that are wasted in traditional data centers hosting production applications by consolidating tasks with lower QoS and SLA requirements. However, the dramatic fluctuation of workloads with lower QoS and SLA requirements may impact the performance of production applications. Frequent task eviction, killing and rescheduling operations also waste CPU cycles and create overhead. This paper aims to schedule hybrid workloads in the cloud data center to reduce task failures and increase resource utilization. The multi-prediction model, including the ARMA model and the feedback based online AR model, is used to predict the current and the future resource availability. Decision to accept or reject a new task is based on the available resources and task properties. Evaluations show that the scheduler can reduce the host overload and failed tasks by nearly 70%, and increase effective resource utilization by more than 65%. The task delay performance degradation is also acceptable.

Keywords Cloud data center · Hybrid workloads · Task scheduling · Multi-prediction · ARMA model · Feedback based online AR model

1 Introduction

Data centers have gained significant popularity as a cost-effective platform. However, in traditional data centers, a tremendous amount of resources are over provisioned to production applications to accommodate fluctuating workloads and peak demands as they have high Service Level Agreement (SLA) requirements and are of the most importance to the enterprise. Typically, one application is deployed on a set of hosts to avoid interference and shortage of resource provision. Although hosts in data centers are usually not idle, most of the time, hosts operate at 10–50% of their full capacity, leading to extra expenses on over-provisioning [1, 2]. As a result, efficient use of data center resources is an important cost factor for many organizations [3]. Cloud computing can solve the problem by intelligently consolidating other tasks with lower Quality of Service (QoS) and SLA requirements and leverage the over-provisioned resource effectively. The data center can then fulfill diverse resource demands and performance objectives of hybrid workloads with high scalability and flexibility.

In such cloud data center with hybrid workloads, production workload should have the highest priority since they are of the most importance to the enterprise. They are latency-sensitive and have the highest QoS and SLA requirements, and should not be killed due to host overload or evicted by other tasks. Other tasks with lower QoS and SLA requirements can also be divided into two categories. Workloads, like non-interactive batch jobs used for computing purposes and enterprise daily management transactional applications, only have deadline constraints or completion time constraints, and are not too sensitive to latency. They can be killed when the host is overloaded. These workloads, called middle workloads in this paper, have lower priorities than production workload. Gratis workloads, like test tasks and free beta applications, are

✉ Haiou Jiang
seagullwill@foxmail.com

Haihong E
ehaihong@bupt.edu.cn

Meina Song
mnsong@bupt.edu.cn

[1] Beijing University of Posts and Telecommunications, Beijing 100876, China

charged substantially less or even not at all. They have the lowest priorities, and can be evicted by tasks with higher priorities or killed once the host is detected overloaded.

Various resource management systems like YARN [4], Mesos [5], and Kubernetes [6] have emerged to manage resource allocation and scheduling, deploy and monitor multiple diverse applications across the entire data center and cloud environment. Typical scheduling algorithms, like Fair Scheduler [7] and Capacity Scheduler [8], are used in these resource management systems based on instantaneous resource availability at the scheduling time. In the cloud data center with hybrid workloads, the task scheduling mechanism brings two challenges. First, when the host resources are fully allocated, the production load increase causes host overload. During the period from detecting the host overload to killing some tasks and releasing resources, the performance of the tasks on the host, especially the production tasks, is already affected. Second, tasks are killed due to the host overload, and tasks with lower priorities are evicted by new tasks with higher priorities if the available resources are not sufficient. These failed tasks waste CPU cycles, and the rescheduling, eviction and killing operations also create overhead.

The Google cluster [9] shows the features of such data center with hybrid workloads. Tasks are typically divided into three priority groups, namely, production, middle, and gratis [10], reflecting the importance of the tasks [11] and the latency requirements [12]. Figure 1 shows a seven-day period of the proportions of the used CPU spent on each event in Google cluster. "Evict" in Google *task event* table means "a task was descheduled because of a higher priority task, or because the scheduler overcommitted and the actual demand exceeded the machine capacity", the same as the eviction and the killing we used in our work. The maximum and average proportion of the used CPU spent on Google "evict" is 28.9% and 11.7%, respectively. Meanwhile, "kill" in Google *task event* table means "a task was cancelled by the user or a driver program, or because another task on which this task was dependent died" [9]. This means that the CPU wasted by evicted and killed task is even higher. Part of Google "fail" and "kill" are caused by user's cancel, which is beyond our consideration. If we exclude the event caused by user's cancel, approximately half of the used CPU is spent on the evicted and killed tasks, and the other half is spent on the successful tasks. As a result, resource waste on the evicted and killed tasks is worthy of consideration.

The scheduling policy must consider load changes to respond gracefully to production workload demand surges. One way to solve this is to estimate the current as well as the future resource availability before scheduling a task. Production, middle and gratis workloads have quite different load patterns. Figure 2 shows a one-day period of the CPU usage of production, middle and gratis tasks of Google cluster measured in each 300-s slot. Production CPU usage is usually stable, making time series of the production workload a stationary process. We use an off-line-trained Auto-Regressive and Moving Average (ARMA) model [13, 14] to predict the production load. CPU usage of middle and gratis tasks is much more irregular and has a much larger peak-to-mean ratio than the production workloads. We use a feedback based online Auto-Regressive (AR) [13, 14] model to predict the host load, which is the sum of CPU usage of the production and the middle and gratis workloads.

In this paper, we present a Multi-Prediction based scheduler for Hybrid Workloads (MPHW) in the cloud data center, so that the task failures and resource waste due to host overload and task eviction is reduced, effective resource utilization of the data center is increased, and the performance of the production tasks is guaranteed. We first construct discrete time series of the production and the host

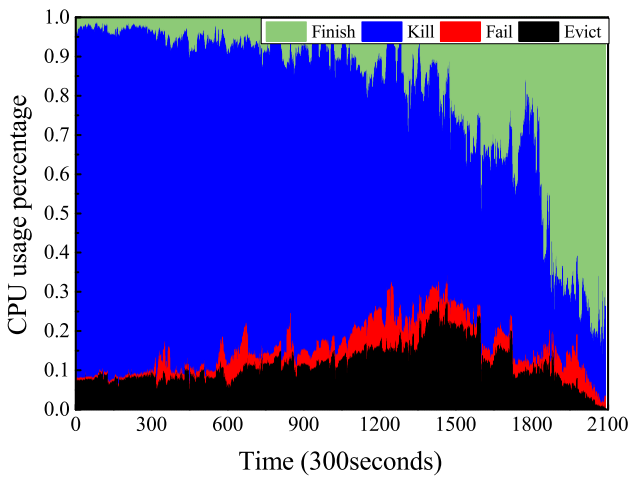


Fig. 1 Proportions of the used CPU spent on each event

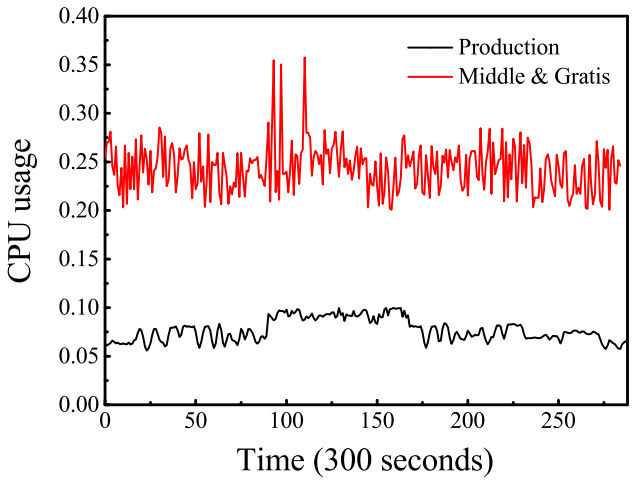


Fig. 2 One-day CPU usage of Google cluster

load. Then, use the offline-trained ARMA model to predict the stationary process of the production load and the feedback based online AR model to predict the time-varying host load. When scheduling a middle task, enough resources should be reserved for future increase of production load. If the available resources are sufficient, the task can be scheduled immediately. Otherwise, it will evict gratis tasks and scheduled. If there is no gratis task, the middle task will be queued and rescheduled until some tasks complete and release resources. When scheduling a gratis task, enough resources should be reserved for future requests of all the pre-scheduled tasks and new tasks with higher priorities during its execution. If available resources are sufficient, the gratis task can be scheduled immediately. Otherwise, it will wait until there are sufficient resources for it. In the experiments, we analyze MPHW by simulating real workload traces from Google cluster. We show that the proposed solution is capable of significantly reducing host overload, task failure, and the CPU waste, complying with SLAs in terms of task scheduling delay and task response time. MPHW fits current resource management systems and schedulers. The multi-prediction model decides how many available resources can be provided to new tasks for diverse workloads. It uses ARMA model for stable load, feedback based online AR model for irregular load, and multi-prediction model for hybrid workloads. It is, thus, a practical scheduler for hybrid workloads with priorities.

This paper is a substantially extended version of our previous short conference paper [15] and improves experiments in several ways. The key improvement, which makes the experiment result more convincing and significantly improves the quality of the work, is extending the scale of the data center and the input workloads. In addition, we add the offline trained ARMA model and the Feedback based Online trained AR (FOAR) model for comparison. Moreover, we add more results analysis, like success rate and MAPE (Mean Absolute Percent Error) for prediction accuracy analysis and effective resource utilization of the data center for performance analysis.

2 Related work

Various resource management systems have emerged to manage hybrid applications in the cloud data center. Apache YARN [4] and Shark [16] used Fair Scheduler [7] and Capacity Scheduler [8], and used priorities as weights to determine the fraction of total resources that each application can get. Mesos [5] used Dominant Resource Fairness (DRF) [17] and guaranteed resources for workloads with higher priorities by restricting dynamic resource sharing and potentially starving other workloads with lower priorities. Google Omega [18] granted each application full access to the entire cluster. Each application can lay claim to the resources using the resilient master copy of the entire state of the cluster in an atomic commit. In Google Kubernetes [6], groups of applications are scheduled in the unit called pod in two steps. The first step is to filter all the nodes under certain requirements of the pod including "No Disk Conflict", "No Volume Zone Conflict", "Host Name", "Max Elastic Block Store Volume", "Check Node Memory Pressure", and "Check Node Disk Pressure" etc. The second is to rank the remaining nodes and find a best fit for the pod. The scheduling algorithms used in these resource management systems are based on instantaneous resource availability at the scheduling time, whereas our scheduler is based on the future resource availability. Meanwhile, priorities are used to determine the upper bound of resources, whereas priorities in our system is used for adopting different scheduling strategies when the resources are not available. Microsoft Apollo [19] scheduled online production services consisting interdependent tasks. It considered the history and the probability of task failure, then schedules tasks to the server minimizing the task completion time based on future resource availability. The purpose of Apollo is to reduce latency of online workflows, whereas our purpose is to reduce resource waste for hybrid workloads consisting independent tasks. Quasar [20] used collaborative filtering techniques to determine the least amount of the resources to meet performance constraints specified by users based on the current state of the cluster. Quasar predicts the resource requirement to increase resource utilization instead of relying on resource reservation, while our work predict future load for hybrid workloads before scheduling.

In the research on hybrid workloads scheduling in the data center, existing studies focus on ensuring the QoS and SLA requirements of each workload. Carrera et al. [21, 22] developed a technique to fairly manage mixed workloads in terms of both batch jobs and transactional applications. Their aim is toward a fairness goal while also trying to maximize individual workload performance, and our aim is to efficiently utilize the data center resources while insuring the performance of tasks with the production workload. Garg et al. [23, 24] considered transactional workloads and non-interactive batch jobs, and used admission control and Virtual Machine (VM) rescheduling to maximize resource utilization and ensure different SLAs of hybrid workloads. The work treats transactional and batch workloads equally, while our paper treats hybrid workloads with different priorities. Garg et al. used Artificial Neural Network (ANN) to predict the future resource availability and the expected resource demand of each application. The ANN forecasting model works well in the context of Grids [25], where the workload shows the feature of a stationary

process, but it can not fit well in an irregular host load in the cloud data center. Curinom et al. [26] introduced reservation-based scheduling for production jobs and best-effort jobs in big-data frameworks, trying to guarantee stringent SLAs for production jobs and minimize latency for best-effort jobs. They formalized planning of current and future cluster resources as a Mixed Integer Linear Programming (MILP) problem and proposed scalable heuristics to balance resource allocation between production jobs and best-effort jobs. While our paper prioritizes production tasks to guarantee their SLAs at the expense of other tasks' latency. Sharma et al. [27] proposed a heterogeneous data center with both interactive and batch workloads, as well as both virtual and native machines in the data center. They utilize available unused resources by consolidating middle jobs with over-provisioned foreground applications to improve application performance and energy efficiency. They used an interference prevention system to monitor interference and killed tasks after the occurrence of interference. The afterward correction only works after problems happen and cannot effectively prevent serious performance degradation. We predict the host load before making scheduling strategies so that it can detect potential risks caused by scheduling new tasks to the host and prevent performance degradation.

Studies on host load prediction usually attempt to provide benchmarks for virtual machine migration, server consolidation and energy management. Farahat et al. [28] used a Curve Fitting Prediction (CFP) technique combined with Genetic Algorithms (GAs) to obtain the optimum parameters of a Gaussian prediction model. It provides very accurate hourly load forecast, but the overhead of the prediction is too high for real-time task scheduling. Khan et al. [29] grouped VM first, and then designed a model to capture the CPU load of different groups by leveraging the Hidden Markov Model (HMM). Yang et al. [30] combined the Phase Space Reconstruction (PSR) method and the Group Method of Data Handling (GMDH) based on the Evolutionary Algorithm (EA) to predict not only the mean load in consecutive future time intervals but also the actual load in each consecutive future time interval. Yang et al. [31] proposed a new multi-step-ahead prediction approach for CPU load that is more accurate than repeating the one-step-ahead prediction approach in four steps: finding a fit function for the change range sequence, predicting the change pattern, composing the change range, and changing the pattern prediction. Di et al. [32, 33] used a Bayes model to change the prediction process into a classification problem, identified novel predictive features of the host load, and predicted the mean load over a long-term time interval. The problem with the Bayes model is that the length of the prediction time interval increases exponentially. With the growth of the segment length, the mean load could not fully reflect the fluctuation of the host. Zhang et al. [34] used the offline-trained ARIMA model to predict the load over a time window and achieved multi-step prediction by iterating the one-step prediction to minimize the total energy cost while meeting the performance objective in terms of task scheduling delay. The above methods can either achieve good accuracy or multi-step prediction. However, the prediction methods are based on offline training and parameters or patterns of the model are all static and fixed in the prediction process. They work well for workload that shows features of a stationary process or has a fixed change pattern, but they cannot fit well for rapidly changing workloads with irregular patterns. In our work, we combine the offline-trained ARMA model to predict the stationary production load for simplicity and the feedback based online AR model to predict the irregular host load for accuracy.

3 Hybrid workloads scheduling strategy and system architecture

In this paper, we present a cloud data center containing hybrid workloads. The scheduler estimates future as well as the current resource availability. For simplicity, the scheduler is based on the static task placement, and does not save the task progress, neither consider live migration. It terminates the task to release resources immediately and reschedules it later. This simple killing and eviction policy is widely used by Apache Hadoop [35], Yarn [7], Google Borg [36], Omega [18], and Quasar [20]. Tasks are supposed independent so that killing and eviction of tasks does not impact the communication partners.

Hybrid workloads are typically divided into three categories according to their priorities. Scheduling strategy is made according task priorities and prediction is made according to load patterns.

Production workloads are the original tasks and jobs hosted on the servers and have the highest priorities. They are latency sensitive and have the highest QoS and SLA requirements. A new production task should be scheduled as soon as it comes into the data center, and it will evict middle or gratis tasks if the available resources are not sufficient.

Middle workloads, such as batch jobs and enterprise management transactional applications, have middle priorities. If available resources is sufficient for a new middle task, the task can be scheduled immediately. Otherwise, it will evict gratis tasks and scheduled, or queued if there is no gratis task. During the execution of the middle task, if the production workload requires more resources and there are not sufficient resources, performance of the production workload will be affected and some tasks will be killed.

Enough resources should be reserved for future requests from the production workload during the execution of the middle task. We predict the production workload, and the available resources are the current spare resources subtracting the future resource request from the production workload.

Gratis workloads, such as test tasks and free beta applications, have the lowest priorities. They can be evicted by tasks with higher priorities or killed once the host is detected overloaded. If the available resources are not sufficient for a new gratis task, it will be queued until the resources are released by other tasks. During the execution of the gratis task, if the spare resources are not sufficient, the increase of resource usage increase from the pre-scheduled tasks will cause performance degradation, and the new task submission with higher priorities will evict some gratis tasks. Sufficient resources should be reserved for future resource requests from both the production workload and new middle tasks during the execution of the gratis task. We predict the total host load, and the available resources are the current spare resources subtracting the future host load increase.

The proposed hybrid workloads scheduling system architecture is depicted in Fig. 3. It consists of the following components:

- *Task classifier* identifies the category of the input task and imports different predictors for it. For a middle task, an offline predictor is used. For a gratis task, an online predictor is used.
- *Offline predictor* uses the ARMA model with static parameters obtained from offline training. It predicts the maximum resource request of the production workloads in the next prediction window. Then, it suggests the maximum amount of resources that can be safely allocated to the new task.
- *Online predictor* uses the feedback based online AR model with parameters dynamically updated. It predicts the maximum host load in the next prediction window. Then, it suggests the maximum amount of resources that can be safely allocated to the new task.
- *Scheduler* decides the scheduling operation for the new task according to its priority and the suggested available resources given by the predictors. If the suggested available resources are not sufficient for the task, a middle task will evict gratis tasks and be scheduled, or queued if there is no gratis task; a gratis task will be queued and rescheduled until some tasks complete and release sufficient resources.
- *Monitor and management module* is a background program running periodically to check total resource utilization of each host. Once a host is overloaded, the module will kill some tasks to release resources for production applications.
- *Task waiting Queue* contains killed and evicted tasks. These tasks are sorted based on deadlines up to which the tasks must be scheduled and executed successfully. The deadline is calculated as the latest task finish time subtracting the task completion time. If the available resources are enough for the tasks in the queue, they will be submitted and scheduled. Tasks that reach the deadlines can be scheduled to other data centers if available resources are still not sufficient. In this paper, we focus on task scheduling in one data center and do not discuss the operations of scheduling tasks to other data centers. We assume that all of the tasks do not have deadlines and can be scheduled and finished eventually at spare time.

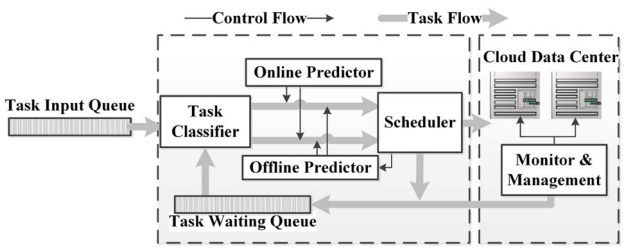


Fig. 3 Hybrid workloads scheduling system architecture

The proposed MPHW scheduler is depicted in Algorithm 1. When a task arrives, it first identifies the category and imports different predictors for the task. For a middle task, the ARMA model is used to predict the production load and available amount of resources of each host (Line 4). Then chooses the host h_j with the maximum available resources a_j (Line 6). If h_j provides all the resources r_i required by t_i, schedules t_i to h_j (Line 7–8). Otherwise, it chooses one host h_k that can provide enough resources to t_i by evicting tasks with lower priorities on the host. h_k has the potential resources p_k larger than r_i, where p_k is the sum of a_k and resource requests of tasks with lower priorities than t_i on h_k. In this paper, the target host h_k is chosen randomly for simplicity (Line 12). Then, chooses victim tasks with lower priorities until the new available resources a'_k is equal to or larger than r_i, where a'_k is the sum of a_k and the resource requests of the victim tasks (Line 13). Then, evicts the victim tasks and schedules t_i to h_k (Line 14). Tasks with the lowest priority should be chosen first. Strategies to choose victim tasks with the same priorities are the following: (1) least fit, evicting tasks with the smallest resource requests; (2) best fit, evicting tasks to get the minimum a'_k while $a'_k > r_i$; (3) worst fit, evicting tasks with the largest resource requests; (4) least execution process first, evicting tasks with the least execution process; (5) maximum finish time first, evicting tasks with the longest finish time; (6) latest start time first, evicting tasks

with the latest start time, etc. For simplicity, the victim tasks are chosen randomly (Line 13) in this paper. If there is no host with p_k larger than r_i, the task is placed into the waiting queue (Line 28) and rescheduled when some tasks finish and release sufficient resources. For a gratis task, the scheduler first recalculates the parameters of the AR(p) model (Line 21), then predicts the total workload and the available resource of each host using the AR(p) model (Line 22), then chooses the host with maximum available resources a_j (Line 24). If h_j provides all the resources r_i required by t_i, schedules t_i to h_j (Line 25–26). Otherwise, the task is placed into the waiting queue (Line 28) and rescheduled when some tasks finish and release sufficient resources. There is also a monitor program running periodically to detect host overload. Once a host is overloaded, the monitor chooses and evicts victim tasks until overload is eliminated.

Algorithm 1

Pseudo code of the multi-prediction based scheduling algorithm for hybrid tasks

```
Input: T = {t_1, t_2, ..., t_n}: Queue of incoming tasks
       WQ = {t_w1, t_w2, ..., t_wn}: Queue of waiting tasks
       H = {h_1, h_2, ..., h_n}: Set of hosts in the data center
 1:  for Each task t_i in T and WQ do
 2:   if t_i is middle task do
 3:     for Each host h_j in H
 4:       Use ARMA model to predict available resources a_j
 5:     end for
 6:     Choose the host h_j with maximum a_j
 7:     if a_j > r_i do
 8:       Schedule t_i to h_j
 9:     end if
10:     else do
11:       if There is at least one target host do
12:         Choose a target host h_k randomly
13:         Choose victim tasks in h_k randomly until a'_k > r_i
14:         Evict the victim tasks and schedule t_i to h_k
15:       end if
16:       else Put t_i into waiting queue WQ
17:     end else
18:   end if
19:   if t_i is gratis task do
20:     for Each host h_j in H
21:       Update parameters of AR(p) model
22:       Use AR(p) model to predict available resources a_j
23:     end for
24:     Choose the host h_j with maximum a_j
25:     if a_j > r_i do
26:       Schedule t_i to h_j
27:     end if
28:     else Put t_i into waiting queue WQ
29:   end if
30:  end for
```

4 Multi-prediction of hybrid workloads

Load is a continuous function of time. We divide continuous time into discrete time slots. The maximum value of the load in the time slot is the value of time series. Then, we obtain the time series $\{X_n\}$ of the load. We use the ARMA model to predict the stationary process of the production load. Parameters are trained off line.

For the host load, the sum of the resource usage of the production, middle and gratis workload, the time series of the load shows evidence of non-stationarity. The parameters of the AR model are updated dynamically based on feedback to meet the continuously changing pattern of time-varying load series.

4.1 ARMA prediction model

In the statistical analysis of time series, the ARMA model provides a parsimonious description of a weakly stationary stochastic process consisting of the AR process and the Moving Average (MA) process [37]. Given that the time series $\{X_n\}$ satisfies the ARMA(p, q) [13, 14] model and the previous $n-1$ values $\{X_{n-1}\}$ are known, the expected value X_n at time n is formulated as follows:

$$X_n - \varphi_1 X_{n-1} - \cdots - \varphi_p X_{n-p} = \varepsilon_n - \theta_1 \varepsilon_{n-1} - \cdots - \theta_q \varepsilon_{n-q}, \quad (1)$$

where X_t, $t = n-p, n-p+1, \ldots, n$ is the value of time series at time t. p is the order of AR process and q is the order of the MA process. φ_i, $i = 1, 2, \cdots, p$ and θ_j, $j = 1, 2, \ldots, q$ are parameters estimated from the previous $n-1$ values $\{X_{n-1}\}$. ε_k, $k = n-q, n-q+1, \ldots, n$ are error terms that are generally assumed to be white Gaussian noise, namely,

$$E(\varepsilon_n) = 0, \quad E(\varepsilon_n \varepsilon_{n+k}) = \begin{cases} \sigma_\varepsilon^2 & k = 0 \\ 0 & k \neq 0. \end{cases} \quad (2)$$

It is generally considered good practice to find the smallest values of p and q that provide an acceptable fit to the data. Finding the appropriate values of p and q in the ARMA(p, q) model can be facilitated using Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC) [13]. AIC and BIC are based on information theory and provide scores of the quality of a model with regard to the parameters that are selected by the maximum likelihood method. By definition, AIC can be described as the following:

$$\text{AIC} = 2 \times (\textit{number of parameters}) - 2 \times \ln(\textit{maximum likelihood}), \quad (3)$$

and BIC can be described as the following:

$$\text{BIC} = 2 \ln(\textit{number of data item}) \times (\textit{number of parameters}) - 2 \times \ln(\textit{maximum likelihood}). \quad (4)$$

A lower value of AIC and BIC indicates a better model.

ARMA models in general can, after choosing p and q, be fitted by least squares regression to find the values of the parameters that minimize the error term. For the training set of the time series satisfying the ARMA(p,q) model, the

vector of the parameters is donated as $\bar{\boldsymbol{\varphi}} = (\varphi_1, \ldots, \varphi_p, \theta_1, \ldots, \theta_q)^{\mathrm{T}}$. The estimated value of ε_n is donated as $\hat{\varepsilon}_n$. It is calculated recursively as follows:

$$\hat{\varepsilon}_n = \begin{cases} 0, & n \le p \\ x_n - \sum_{i=1}^{p} \varphi_i x_{n-i} + \sum_{i=1}^{q} \theta_i \hat{\varepsilon}_{n-i}, & n = p+1, \ldots, N. \end{cases} \tag{5}$$

We define the quadratic sum of $\hat{\varepsilon}_n$ as $S(\bar{\boldsymbol{\varphi}})$, which is formed as

$$S(\bar{\boldsymbol{\varphi}}) = \sum_{n=p+1}^{N} \hat{\varepsilon}_n^2. \tag{6}$$

The value $\hat{\bar{\boldsymbol{\varphi}}}^L = (\hat{\varphi}_1^L, \ldots, \hat{\varphi}_p^L, \hat{\theta}_1^L, \ldots, \hat{\theta}_q^L)$ that allows $S(\bar{\boldsymbol{\varphi}})$ to obtain the minimum is the approximate value of the parameters of the ARMA(p,q) model.

4.2 Feedback based online AR prediction model

The ARMA(p,q) model consists of an AR(p) model and a MA(q) model, where the AR(p) model is in the following form:

$$X_n = \varphi_1 X_{n-1} + \varphi_2 X_{n-2} + \cdots + \varphi_p X_{n-p} + \varepsilon_n, \tag{7}$$

meaning that X_n is calculated by the linear combination of previous p values. The MA(q) model is in the following form:

$$X_n = \varepsilon_n - \theta_1 \varepsilon_{n-1} - \theta_2 \varepsilon_{n-2} - \cdots - \theta_q \varepsilon_{n-q}, \tag{8}$$

meaning that X_n is calculated by the linear combination of previous q prediction errors.

The host load is irregular and the prediction accuracy will decrease, meaning that the error term ε_k, $k = 1, 2, \ldots, n-1$ in the MA(q) model will be large. This makes the ARMA model even less accurate. Thus, we only use the AR(p) model for the host load prediction. In the meantime, fixed parameters do not satisfy the continuously changing pattern of the time series. We recalculate parameters dynamically based on the real load feedback each time before using the AR prediction model.

There are many ways to estimate the parameters, such as the ordinary least squares procedure, method of moments through Yule-Walker equations, and Markov chain Monte Carlo methods [13]. We use Yule-Walker equations as follows:

$$\begin{cases} \rho_1 = \varphi_1 + \varphi_2 \rho_1 + \cdots + \varphi_p \rho_{p-1} \\ \rho_2 = \varphi_1 \rho_1 + \varphi_2 + \cdots + \varphi_p \rho_{p-2} \\ \qquad \vdots \\ \rho_p = \varphi_1 \rho_{p-1} + \varphi_2 \rho_{p-2} + \cdots + \varphi_p, \end{cases} \tag{9}$$

where ρ_k is the auto-correlative function and calculated as follows:

$$\rho_k = \rho_{-k} = \frac{\gamma_k}{\gamma_0}, \quad k \ge 0, \tag{10}$$

where γ_k is the auto-covariance and calculated as follows:

$$\gamma_k = \gamma_{-k} = E[(X_n - \mu)(X_{n-k} - \mu)], \quad k \ge 0. \tag{11}$$

In particular,

$$\gamma_0 = E[X_n X_n] - E[X_n] E[X_n] = \sigma^2 - \mu^2. \tag{12}$$

According to AIC and BIC, the order p equals 2, meaning that the AR(2) model in this paper is simple and accurate. From the Yule–Walker equation, we have the following:

$$\begin{aligned} \varphi_1 &= \frac{\rho_1(1-\rho_2)}{1-\rho_1^2} = \frac{r_0 r_1 - r_1 r_2}{r_0^2 - r_1^2}, \\ \varphi_2 &= \frac{\rho_2 - \rho_1^2}{1-\rho_1^2} = \frac{r_0 r_2 - r_1^2}{r_0^2 - r_1^2}. \end{aligned} \tag{13}$$

The time series of the host total workload show evidence of non-stationarity. An initial differencing step is applied to remove the non-stationarity. The first-order differenced time series $\{\Delta X_n\}$ of the AR(p) model is in the following form:

$$\Delta X_n = \varphi_1 \Delta X_{n-1} + \varphi_2 \Delta X_{n-2} + \cdots + \varphi_p \Delta X_{n-p} + \varepsilon_n. \tag{14}$$

For the first order differenced AR(2) model,

$$X_n - X_{n-1} = \varphi_1 (X_{n-1} - X_{n-2}) + \varphi_2 (X_{n-2} - X_{n-3}) + \varepsilon_n. \tag{15}$$

Thus, we obtain the following equation for the host total workload prediction:

$$X_n = (1 + \varphi_1) X_{n-1} + (\varphi_2 - \varphi_1) X_{n-2} - \varphi_2 X_{n-3} + \varepsilon_n. \tag{16}$$

Each time we obtain a new workload record X_n, we recalculate parameters φ_1 and φ_2 to make the prediction model constantly adjust to the changing pattern of workload.

5 Experimental evaluation

In this section, we conduct trace-driven simulations to realistically evaluate the performance improvement by using multi-prediction based scheduling for hybrid tasks. The experiments include prediction accuracy analysis, task scheduling performance analysis and comparisons of different slot sizes.

5.1 Experimental settings

We use the real-world workload traces of Google cluster [9] to construct the hybrid workloads. Submission time, priorities, and CPU, memory, and disk request of tasks are recorded in the Google *task event* table. Tasks with priorities larger than 8 are the production tasks, tasks with priorities smaller than 2 are the gratis tasks, and the others are the middle tasks [10]. The CPU, memory, and disk request are normalized to 1. The slot as the prediction time window and the measurement unit is set to 300 s consistent with the measurement period of Google cluster [38]. All the tasks submitted in the first seven days (around 2100 slots) in the Google cluster are used in the experiment. There are about 11.5 million tasks used in the experiment.

The load is the sum of CPU request of the tasks executed at the same time and load in one slot is the maximum value in the slot. Figure 4 shows the load of the input production tasks and the middle and gratis tasks in the cloud data center of the 2100 slots. Most of the time, load pattern of production workload is stable and large spikes happen from slot = 600 to slot = 700, and at slot = 900. Load pattern of batch and gratis tasks is much irregular and large spikes happen from slot = 650 to slot = 700, with the sum of CPU request even larger than 1000. A spike means that a lot of tasks are submitted to the data center concurrently. This may cause resource contention, leading to task eviction and delay scheduling. Meanwhile, performance of production workload may be affected.

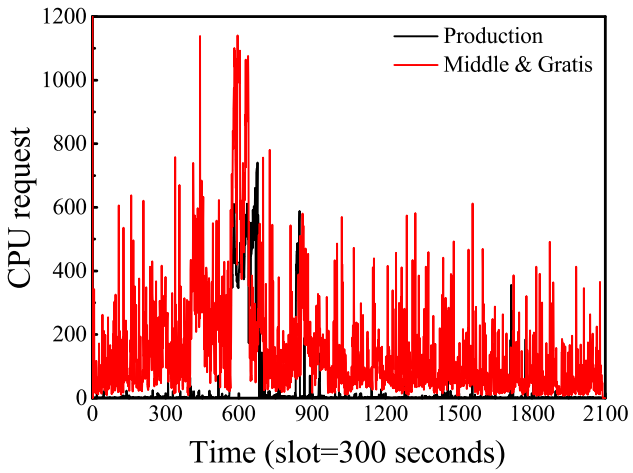


Fig. 4 Load of the input hybrid workloads in the cloud data center

We use CloudSim 3.0 [39] to simulate the cloud data center. The data center consists of 1000 hosts, with CPU, memory, and disk capacity normalized to 1. A background program monitors each host periodically. Once a host is overloaded, tasks with the lowest priorities are killed and the occupied CPU resources are released.

We compare the task scheduling performance of four different scheduling methods:

- *Original* method does not distinguish task priorities and different load patterns of the hybrid workloads. It makes scheduling decisions based on the current available resources.
- *ARMA* does not distinguish task priorities and different load patterns of the hybrid workloads. It uses the ARMA model to predict the host load and available resources before scheduling any type of task.
- *FOAR* does not distinguish task priorities and different load patterns of the hybrid workloads either. It uses the Feedback based Online AR model to predict the host load and available resources before scheduling any type of task.
- *MPHW* is the Multi-Prediction based scheduling for Hybrid Workloads, which considers both task priorities and different load patterns of hybrid workloads.

5.2 Prediction accuracy analysis

One of the challenging issues of the scheduler is to determine the suitable amount of available resources for a new task. Multiple prediction models are used for hybrid workloads with different load patterns. The first experiment is to evaluate the prediction accuracy.

The predictive study uses the production and the host load trace generated by the original scheduler. The ARMA model is used to predict the production load and the host load separately. For the ARMA model, we split the 7-day trace data into two durations, a training period, the first 300 slots, and a validation period, the last 1800 slots. The training period is used to fit the model and the validation period is used to validate the prediction accuracy. We use the IBM SPSS Statistics [40] toolkit to analyze the training set and validate that the production load satisfies the stationary process, then obtain parameters $p = 1$, $q = 1$, and coefficients φ_1 and θ_1. The online AR model is used to predict the host load. Coefficients φ_1, φ_2 are updated online based on feedback data. The model does not have an offline-training step, we use the data of the last 1800 slots as the validation set consistent with the ARMA model.

We use Success Rate and Mean Absolute Percent Error (MAPE) to evaluate the prediction accuracy. The Success

Rate is the ratio of the number of successful predictions to the total number of predictions. Di et al. [32, 33] defined that the prediction is a success if it falls within 10% of the real value. The predicted value lower than the real value means that the predicted amount of available resources is larger than the real amount. This may cause resource over provision. Therefore, the situation that the predicted value is lower than the real value is defined as a failure. We define the prediction as a success if it is within 10% larger than the real value, which means,

$$\frac{\hat{X}_i - X_i}{X_i} \times 100\% \leq 10\%. \tag{17}$$

MAPE is calculated as follows:

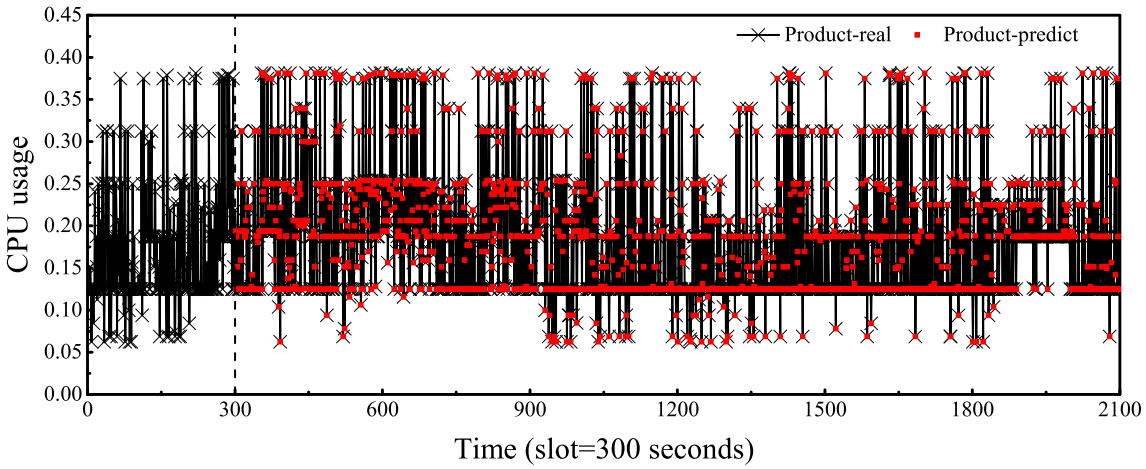
$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{\hat{X}_i - X_i}{X_i}\right| \times 100\%. \tag{18}$$

In Eqs. (17) and (18), X_i is the real value, $\hat{X}_i$ is the predicted value, N is the number of predicted values. In general, a higher Success Rate and lower MAPE means a better the prediction.

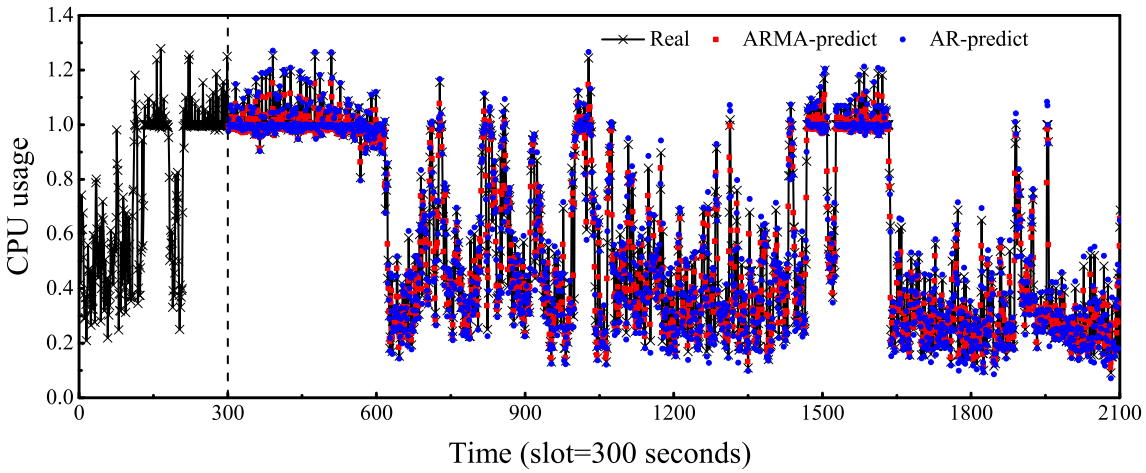
We randomly choose one host and draw plots of the real and the predicted load in Fig. 5. The Cumulative Distribution Function (CDF) of the Success Rate and MAPE of the predicted load of the 1000 hosts in the 1800 slots are shown in Fig. 6.

Figure 5a shows the real and the predicted production load using the ARMA model. The predicted values are quite close to the actual values. The average success rate is 71.8%, and MAPE is 1.81%. It shows that the offline trained ARMA model is accurate enough for the stationary process of the production load.

Figure 5b shows the real and the predicted host load using the ARMA model and the feedback based online AR model separately. The reason why the resource usage

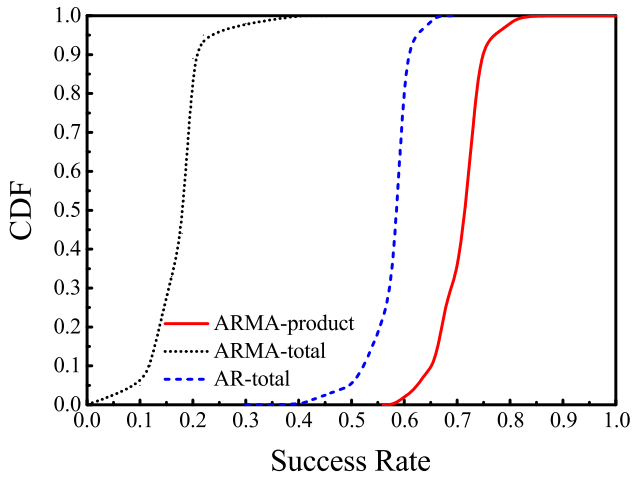


(a) ARMA prediction of the production load.

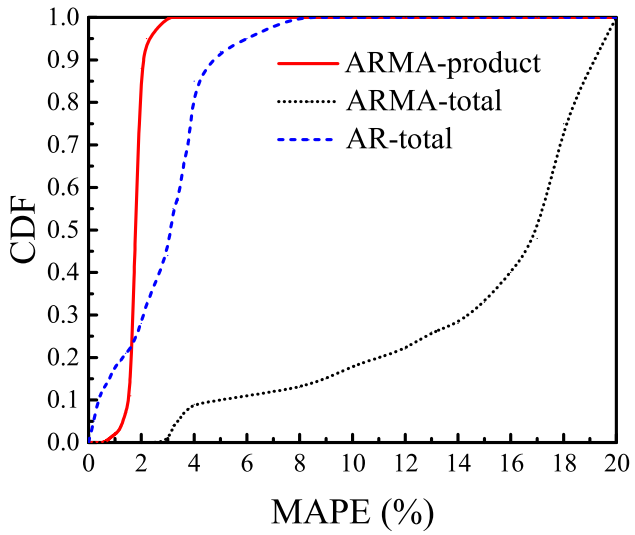


(b) ARMA and AR prediction of the host load.

Fig. 5 Real and predicted load



(a) Success Rate of prediction.



(b) MAPE of prediction.

Fig. 6 CDF for different prediction methods

exceeds 100% is because the host is overloaded by resource over provision. The prediction plots show that the feedback based online AR model is closer to the real values. The Success Rate and MAPE in Fig. 6 shows that the feedback based online AR model performs better than the ARMA model. The average Success Rates are 17.22% of the ARMA model and 58.3% of the AR model. The MAPEs are 15.14% of the ARMA model and 2.63% of the AR model. The feedback based online AR model is accurate for the time-varying host load.

5.3 Data center performance improvement

In this section, we compare the performance improvement of the cloud data center using the four methods.

5.3.1 Task failure and scheduling decrease

The number of instances of host overload, task killing and task eviction in the data center of the four methods is shown in Fig. 7. The instances of host overload decrease from 3.03% to 2.40% of ARMA, 1.19% of FOAR and 1.38% of MPHW, showing an improvement of 20.0%, 60.3%, and 54.85%, respectively. When the host is overloaded, all the tasks running on the host are slowed down and the performance is impacted. The MPHW has the highest performance insurance.

The fraction of task failures, including killed tasks and evicted tasks, has also decreased, from 19.11% to 8.94% of ARMA, 7.21% of FOAR and 5.74% of MPHW, which is about 53.2%, 62.3%, and 69.92% decrease, respectively. A task is resubmitted to the data center when it becomes runnable after failure, including killing and eviction, and

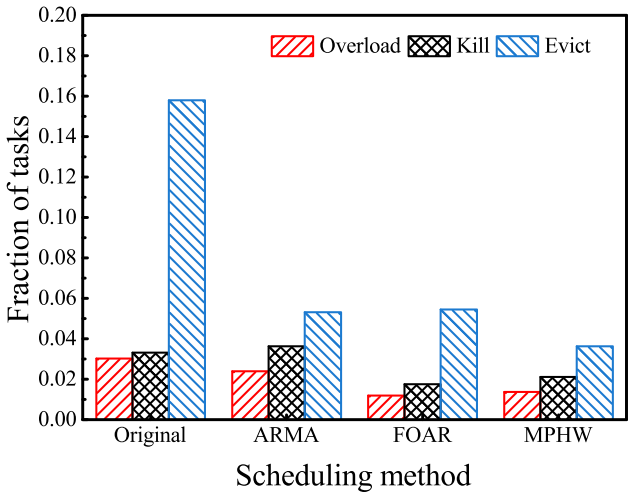


Fig. 7 Number of host overload, task killing, task eviction in the data center

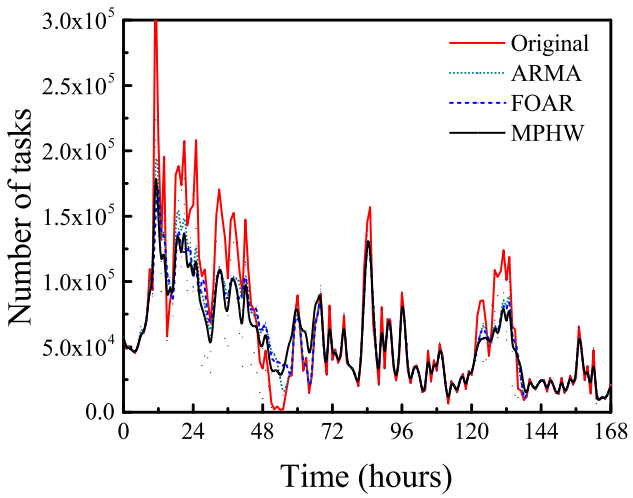


Fig. 8 Task reschedule comparison

rescheduled when there are available resources. With the fraction of task failure decreased greatly, tasks resubmission are reduced as well. Figure 8 shows the number of the overall scheduled tasks, including first submitted tasks and rescheduled tasks, measured in each hour. The original method has large spikes in the task scheduling, while ARMA, FOAR, and MPHW method have smooth task scheduling. ARMA has 8.55% fewer scheduled tasks than the original method, and FOAR has 10.01% fewer scheduled tasks than the original method. MPHW has 11.22%, 2.92%, and 1.34% fewer scheduled tasks than the original method, ARMA, and FOAR, respectively. Large spikes of scheduling means that many tasks are scheduled at the same time. This may cause resource contention and task eviction.

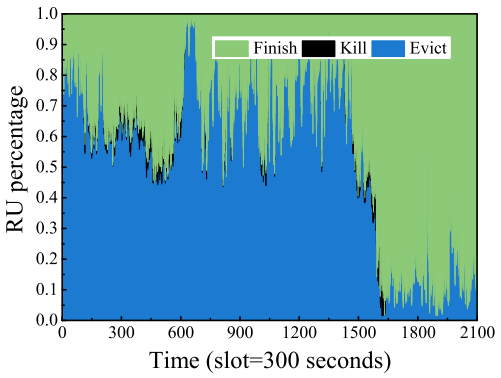
5.3.2 Effective resource utilization increase

The failed tasks waste CPU cycles, and the rescheduling, eviction and killing operation also create overhead. In this section, we will show the increase of effective resource utilization due to the resource waste decrease.

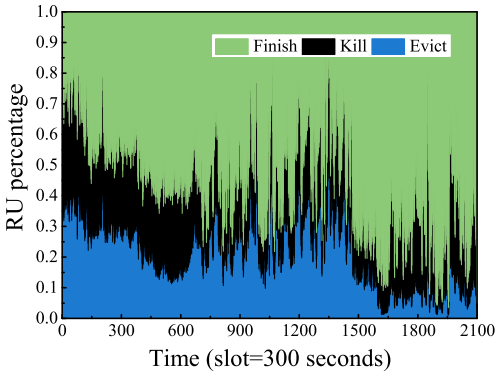
We use the Resource Usage (RU) percentage to compare the resources used by the evicted, killed and finished tasks in the data center. In the slot_j, resource usage RU_j of certain type of task (evicted, killed, finish) is computed as the sum of task CPU usage of the type in the slot_j, as shown in Eq. (19), where CPU_i is the CPU request of task_i. We first compute the total RU of the evicted, killed and finished tasks in the data center. Then, compute the percentage of the RU of each type.

$$\text{RU}_j = \sum_{i \in \{i \mid \text{task}_i \textit{ excute in } \text{slot}_j\}} \text{CPU}_i. \tag{19}$$

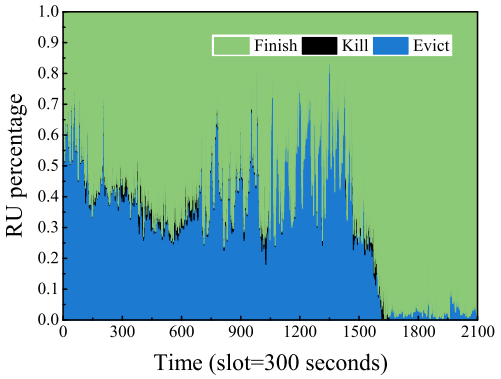
Figure 9 shows the RU percentage of the four methods. The average RU percentage of failed tasks, including killed and evicted tasks, decreases from 59.17% of the original method to 45.24% of ARMA, 38.19% of FOAR, and 29.99% of MPHW. Average RU percentage of tasks finished successfully increases from 40.83% of the original method to 54.76% of ARMA, 61.61% of FOAR, and 70.01% of MPHW. The traditional resource utilization is the CPU usage of the host. However, CPU used for failed tasks are wasted. In this paper, we define the effective resource utilization, measuring how the resources are used effectively. It is the RU percentage of the tasks finished successfully multiplying the total CPU usage. Decrease of killed and evicted tasks saves a great deal of CPU. Figure 10 presents the boxplots showing the minimum, 25%, mean, 75%, and maximum values of the overall CPU used effectively each day. Average effective resource utilizations are 29.10% of the original method, 36.49% of ARMA, 40.57% of FOAR, and 49.12% of MPHW. MPHW



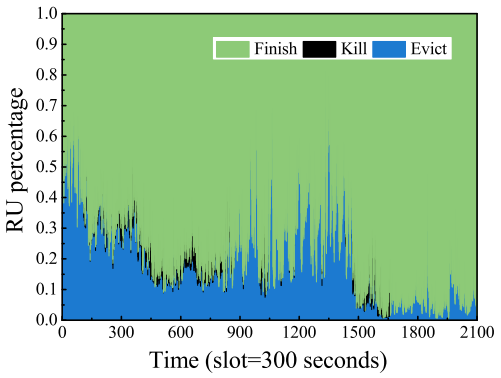
(a) RU percentage of the original method.



(b) RU percentage of ARMA.



(c) RU percentage of FOAR.



(d) RU percentage of MPHW.

Fig. 9 RU percentage of the evicted, killed and finished tasks

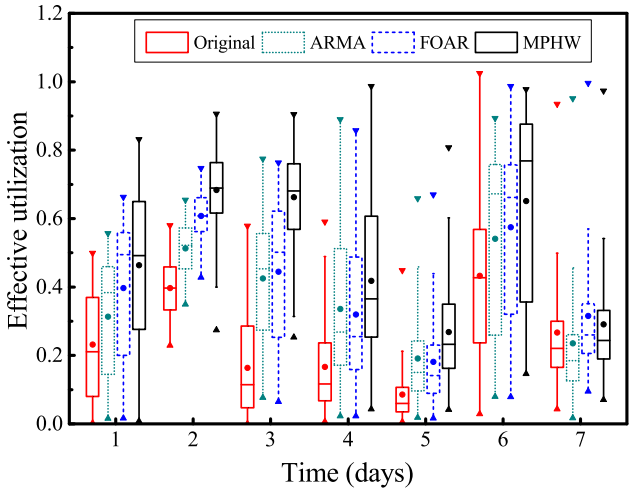


Fig. 10 Effective resource utilization in the data center

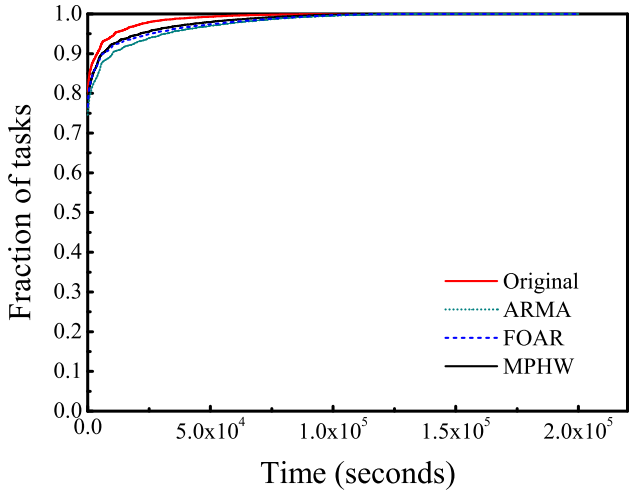
increases effective resource utilization by over 65% than the original method.

5.3.3 Scheduling delay discussion

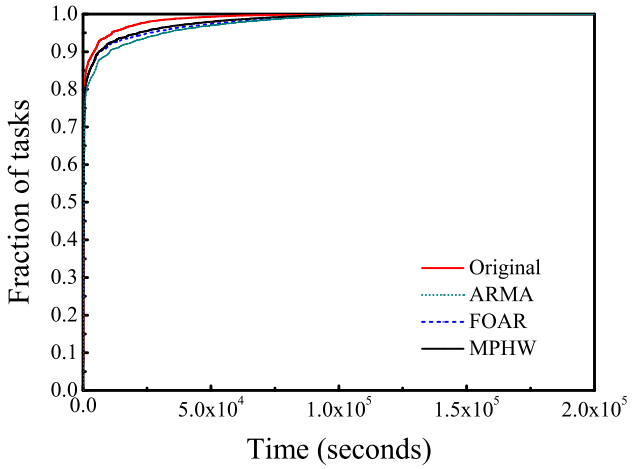
In the prediction based task scheduling, enough resources should be reserved for future resource usage increase from tasks with higher priorities. A task with a lower priority will be delayed with amount of resource request larger than future available resources, instead of scheduled immediately even if the current amount of available resources is larger than requested. This will bring extra scheduling delay. The traditional scheduling delay of a task is the time span from submitted to scheduled. It can not describe the schedule performance correctly if the task is killed or evicted before finished. In this paper, we calculate the task scheduling delay as the time span from the first submitted to the last scheduled before finished successfully. We also calculate the task response time as the time span from the first submitted to the last finished.

Figure 11 shows CDF of scheduling delay and response time. The fraction of tasks scheduled without delay decreases from 79.81% of the original method to 74.74% of ARMA, 76.65% of FOAR, and 77.56% of MPHW. Average scheduling delay increases by 139.78% of ARMA, 101.0% of FOAR, and 75.38% of MPHW. The average response time increases by 122.25% of ARMA, 83.38% of FOAR, and 65.62% of MPHW. An increase of the delay and response time is acceptable because the middle and gratis tasks are insensitive to delay, while the performance of production applications is guaranteed with the peak of the load is diminished, and resource waste is reduced.

The comparison of the original method to the three prediction based methods shows that predicting the amount



(a) CDF of task scheduling delay.



(b) CDF of task response time.

Fig. 11 Task time performance in the cloud data center

of available resources before scheduling can improve the data center performance by reducing task failure, host overload and resource waste. Extra task scheduling delay of the middle and the gratis tasks is acceptable because they have a lower level of QoS and SLA requirements. The comparison of ARMA, FOAR, and MPHW shows that taking into consideration of task priorities and load patterns and using a multi-prediction model leads to greater performance improvement, and smaller task scheduling delay. The comparison of ARMA and FOAR shows that the accuracy of the prediction model impacts the performance of the scheduler.

5.4 Comparison of different slot sizes

HPHW predicts load before scheduling a new task to check whether there are enough resources during its execution time. Thus, the best length of prediction time, referred to

slot in this paper, is approximate to the task execution time. The exact completion time of a new task is difficult to estimate, we use a statistical method. Figure 12 shows the CDF of task completion time. Less than 15% of tasks are finished in 100 s, approximately 60% of tasks are finished in 300 s, approximately 89% of tasks are finished in 600 s, and more than 98% of tasks are finished in 900 s. In this experiment, we compare scheduling performance of different slot sizes to find the relationship between performances and the slot size. The slot size is set to 100 s, 300 s, 600 s, and 900 s separately.

Time is divided into discrete time slots, and the maximum value of load in each slot is selected. With the increase of the slot size, the maximum value increases, making the predicted value larger and the load series less fluctuant. Then, the new task has less possibility of being scheduled, leading to lower host usage and fewer failed tasks. The number of failed tasks, including killed tasks and evicted tasks, of slot = 300 s is 25.57% less than slot = 100 s, and 8.25% and 22.33% more than slot = 600 s, 900 s, respectively. Figure 13 shows the number of the overall scheduled tasks of different slot sizes. With the increase of the slot size, spikes of task submission are smoothing.

With the number of failed tasks decreases, CPU waste also decreases. Effective resource utilization increases, as shown in Fig. 14, from 41.16% of slot = 100 s to 51.87% of slot = 600 s and 58.33% of slot = 900 s.

Figure 15 shows the CDF of task scheduling delay and the response time of different slot sizes. Task time performances of the original method and slot = 100 s are almost the same. With the increase of the slot size, the fraction of delayed tasks decreases from 81.36% of slot = 100 s to 72.04% of slot = 900 s, and the average scheduling delay increases from 2023.84 s of slot = 100 to 3308.30 s of slot = 300 and 7066.61 s of slot = 900 s; the

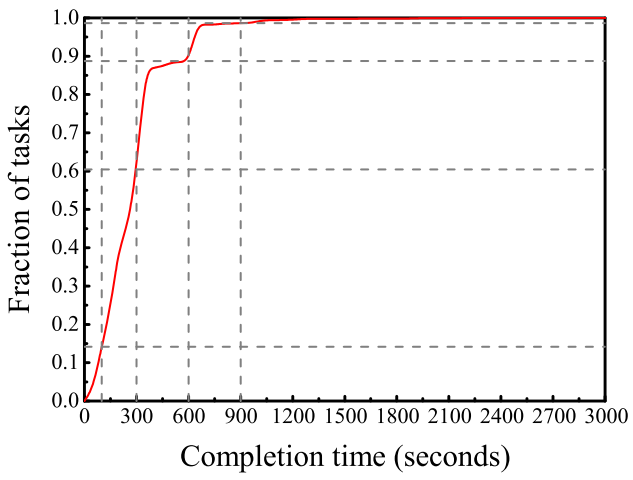


Fig. 12 CDF of task completion time

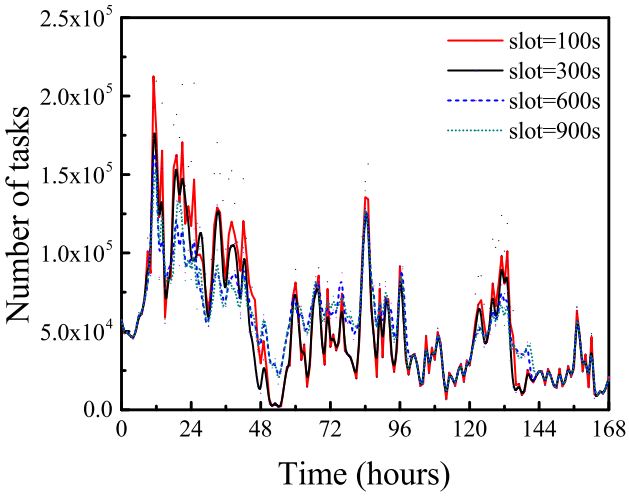


Fig. 13 Task reschedule comparison of different slot sizes

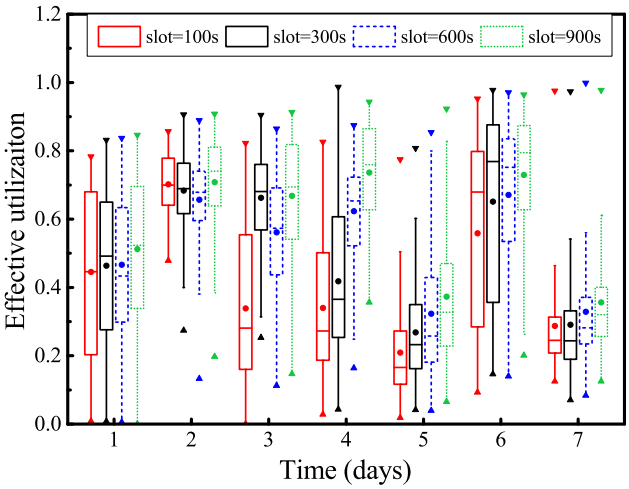
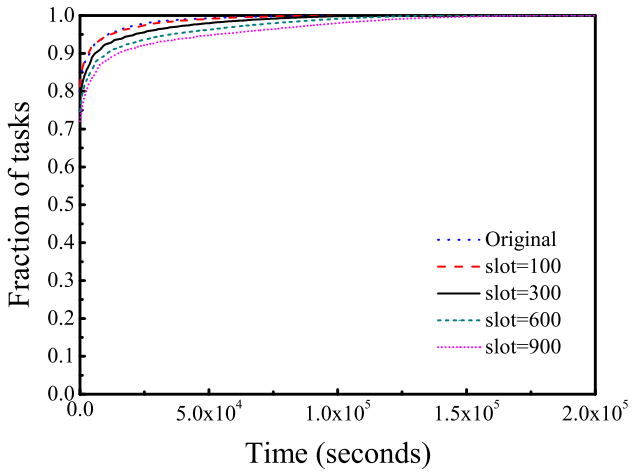


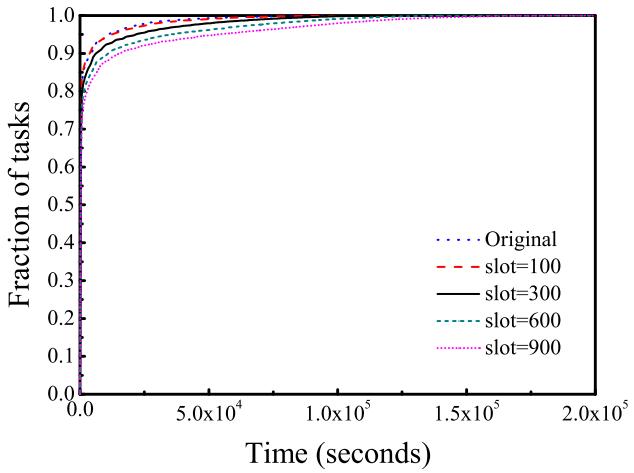
Fig. 14 Comparison of effective resource utilization of different slot sizes

average task response time increases from 2302.74 s of slot = 100 to 3589.31 s of slot = 300 and 7381.74 s of slot = 900 s. The average scheduling delay and response time of slot = 900 s is more than twice as much as slot = 300 s.

The experiments show that the slot size is determined by the completion time of most tasks in the data center, and affects task performance and the resource usage of the data center. If the slot size is too small to cover the completion time of most tasks, the predicted available resources will not reflect real values in the whole execution period of the task. If the slot size can cover completion time of most tasks, with the slot size increasing, task failure decreases and the host effective CPU usage increases. However, scheduling delay performance degrades as new tasks are delayed too long.



(a) Comparison of task scheduling delay.



(b) Comparison of task response time.

Fig. 15 Task time performance comparison of different slot sizes

6 Conclusion

Resources over-provision to production applications in traditional data center causes a waste of resources. Cloud computing can help consolidate middle and gratis tasks with production applications effectively. In this paper, we tackle the challenges of scheduling hybrid workloads by predicting resource availability before scheduling a new task and make different scheduling decisions according to the task priorities. The amount of resources apportioned to a task is subtracted by the future potential load increase during the execution of the new task instead of all the current spare resources to avoid causing host overload and task eviction. We first divide continuous time into discrete slots to construct the time series of the load. Then, we use the ARMA model to predict the stationary process of the production load and the feedback based online AR model to meet the dramatic fluctuations of the host load. For a middle task, if the amount of available resources is not sufficient, it will randomly evict some gratis tasks and be scheduled. For a gratis task, if the amount of available resources is not sufficient, it will be queued until some tasks finish and release sufficient resources. Evaluations show that our multi-prediction based task scheduling policy can reduce the number of instances of host overload and failed tasks by nearly 70% and increase effective resource utilization by more than 65%. Task delay performance degradation is acceptable because the middle and gratis tasks are insensitive to delay. The multi-prediction based scheduling for hybrid workloads can maintain performance of production applications and save computing resources effectively. The multi-prediction model fits current resource management systems and schedulers by using ARMA model for stable workloads, feedback based online AR model for irregular workloads, and multi-prediction model for hybrid workloads. It is, thus, a practical choice for a scheduler for hybrid workloads with priorities.

For the future work, we plan to investigate the interference between hybrid tasks in the same host and find the upper limit of available resources for new tasks while avoiding interference. In this paper, the scheduler is based on the static task placement and uses simple termination, wait, and reschedule policy, live migration will be considered when the available resources are not sufficient in the future work.

Acknowledgements This work is supported by the National Key project of Scientific and Technical Supporting Programs of China (Grant No. 2014BAK15B01); the Cosponsored Project of Beijing Committee of Education; Engineering Research Center of Information Networks, Ministry of Education.

References

1. Beloglazov, A., Buyya, R.: Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput. **24**, 1397–1420 (2012)
2. Guenter, B., Jain, N., Williams, C.: Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In: IEEE INFOCOM, pp. 702–710 (2011)
3. Bhattacharya, A.A., Culler, D., Friedman, E., Ghodsi, A., Shenker, S., Stoica, I.: Hierarchical scheduling for diverse datacenter workloads. In: Proceedings of the 4th Annual Symposium on Cloud Computing, pp. 1–15 (2013)
4. Apache: Yarn. https://hadoop.apache.org/docs/r2.4.1/hadoop-yarn/hadoop-yarn-site/index.html (2013)
5. Apache: Mesos. http://mesos.apache.org/ (2011)
6. Google: Kubernetes. http://kubernetes.io/ (2015)
7. Apache: Fair Scheduler. http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/FairScheduler.html (2016)
8. Apache: Capacity Scheduler. https://hadoop.apache.org/docs/r2.7.0/hadoop-yarn/hadoop-yarn-site/CapacityScheduler.html (2016)
9. Google: Google Cluster Data. http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1 (2011)

10. Reiss, C., Tumanov, A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing, p. 7 (2012)
11. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Towards understanding heterogeneous clouds at scale: Google trace analysis. http://www.pdl.cmu.edu/PDL-FTP/CloudComputing/ISTC-CC-TR-12-101.pdf (2012)
12. Abdul-Rahman, O.A., Aida, K.: Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science, pp. 272–277 (2014)
13. Box, G.E.P., Jenkins, G.M., Reinsel, G.C.: Time series analysis: forecasting and control, 4th edn, pp. 56–81. China Machine Press, Beijing (2011)
14. Hua, L., Qiying, H.: Forecasting and Decision Making, pp. 131–168. China Machine Press, Beijing (2012)
15. Jiang, H., E, H., Song, M.: Hierarchical prediction based task scheduling in hybrid data center. In: 2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS), pp. 17–24 (2014)
16. Apache: Spark. http://spark.apache.org/docs/latest/index.html (2013)
17. Ghodsi, A., Zaharia, M., Hindman, B., Konwinski, A., Shenker, S., Stoica, I.: Dominant resource fairness: fair allocation of multiple resource types. In: Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI), pp. 323–336 (2011)
18. Schwarzkopf, M., Konwinski, A., Abd-El-Malek, M., Wilkes, J.: Omega: flexible, scalable schedulers for large compute clusters. In: European Conference on Computer Systems (EuroSys), pp. 351–364 (2013)
19. Boutin, E., Ekanayake, J., Lin, W., Shi, B., Zhou, J.: Apollo: scalable and coordinated scheduling for cloud-scale computing. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI), pp. 285–300 (2014)
20. Delimitrou, C., Kozyrakis, C.: Quasar: resource-efficient and QoS-aware cluster management. In: Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), pp. 127–144 (2014)
21. Carrera, D., Steinder, M., Whalley, I., Torres, J., Ayguad, E.: Enabling resource sharing between transactional and batch workloads using dynamic application placement. In: Middleware'08, Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware archive, pp. 203–222 (2008)
22. Carrera, D., Steinder, M., Whalley, I., Torres, J., Ayguad, E.: Managing SLAs of heterogeneous workloads using dynamic application placement. In: HPDC'08 2008, Proceedings of the 17th International Symposium on High Performance Distributed Computing, pp. 217–218 (2008)
23. Garg, S.K., Gopalaiyengar, S.K., Buyya, R.: SLA-based resource provisioning for heterogeneous workloads in a virtualized cloud datacenter. In: 11th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP 2011, pp. 371–384 (2011)
24. Garg, S.K., Toosi, A.N., Gopalaiyengar, S.K., Buyya, R.: SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. J. Netw. Comput. Appl. **45**, 108–120 (2014)
25. Dodonov, E., Mello, R.F.: A novel approach for distributed application scheduling based on prediction of communication events. Future Gener. Comput. Syst. **26**, 740–752 (2010)
26. Curinom, C., Difallahu, D.E., Douglasm, C., Krishnanm, S., Ramakrishnanm, R., Raom, S.: Reservation-based scheduling: if you're late don't blame us! In: SOCC '14 Proceedings of the ACM Symposium on Cloud Computing, pp. 1–14 (2014)
27. Sharma, B., Wood, T., Das, C.R.: HybridMR: a hierarchical MapReduce scheduler for hybrid data centers. In: IEEE 33rd International Conference on Distributed Computing Systems, pp. 102–111 (2013)
28. Farahat, M.A., Talaat, M.: Short-term load forecasting using curve fitting prediction optimized by genetic algorithms. Int. J. Enegry Eng. **2**, 23–38 (2012)
29. Khan, A., Yan, X., Tao, S., Nikos, A.: Workload characterization and prediction in the cloud: a multiple time series approach. In: IEEE Network Operations and Management Symposium (NOMS), pp. 1287–1294 (2012)
30. Yang, Q., Peng, C., Yu, Y., Zhao, H., Zhou, Y., Wang, Z., Du, S.: Host load prediction based on PSR and EA-GMDH for cloud computing system. In: IEEE Third International Conference on Cloud and Green Computing, pp. 9–15 (2013)
31. Yang, D., Cao, J., Yu, C., Xiao, J.: A multi-step-ahead CPU load prediction approach in distributed system. In: Second International Conference on Cloud and Green Computing, pp. 206–213 (2012)
32. Di, S., Kondo, D., Cirne, W.: Host load prediction in a Google compute cloud with a Bayesian model. In: Proceedings of the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis, SC, pp. 1–11 (2012)
33. Di, S., Kondo, D., Cirne, W.: Google hostload prediction based on bayesian model with optimized feature combincation. J. Parallel Distrib. Comput. **74**, 1820–1832 (2014)
34. Zhang, Q., Zhani, M.F., Zhang, S., Zhu, Q., Boutaba, R., Hellerstein, J.L.: Dynamic energy-aware capacity provisioning for cloud computing environments. In: ICAC'12 Proceedings of the 9th International Conference on Autonomic Computing, pp. 145–154 (2012)
35. Cheng, L., Zhang, Q., Boutaba, R.: Mitigating the negative impact of preemption on heterogeneous mapreduce workloads. In: CNSM '11, Proceedings of the 7th International Conference on Network and Services Management, pp. 189–197 (2011)
36. Verma, A., Pedrosa, L., Korupolu, M.: Large-scale cluster management at Google with Borg. In: Proceedings of the Tenth European Conference on Computer (Systems EuroSys), p. 18 (2015)
37. ARMA: Auto-Regressive and Moving Average. https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model (2015)
38. Reiss, C., Wilkes, J.: Google cluster-usage traces: format + schema (version of 2011.10.27, for trace version 2). http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1 (2011)
39. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**, 23–50 (2011)
40. IBM: IBM SPSS Statistics. http://www.spss.co.in/ (2014)

Haiou Jiang is currently pursuing Ph.D. in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a B.E. and a M.E. degree from College of Computer Science at Beijing University of Posts and Telecommunications. She is interested in cloudcomputing technology, cloud data center management, service computing.

Haihong E is currently an associate professor in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a B.E. and a M.E. degree from College of Electronic Engineering, a Ph.D. from College of Computer Science at Beijing University of Posts and Telecommunications. She is now working in PCN&CAD center of Beijing University of Posts and Telecommunications. Her research interests are distributed system, SSME (service science, management and engineering), internet of things, wireless city applications, and mobile internet applications.

Meina Song is currently a professor in College of Computer Science at Beijing University of Posts and Telecommunications. She holds a M.E. and a Ph.D. degree from College of Electronic Engineering at Beijing University of Posts and Telecommunications. She is now the chairman of PCN&CAD center of Beijing University of Posts and Telecommunications, vice president of mobile internet application and terminal technology work committee of China communications standards association, member of technical committee of service computing in China computer federation, member of technical committee of network and data communications in China computer federation. Her main research areas include distributed system, P2P technology, service computing, big data technology etc.