Hierarchical Prediction based Task Scheduling in Hybrid Data Center

Haiou Jiang, Haihong E, Meina Song
Beijing University of Posts and Telecommunications, Beijing 100876, China
Email: {seagullwill, ehaihong, mnsong}@bupt.edu.cn

***Abstract*—Cloud computing can help data center consolidate batch and gratis tasks with over-provisioned production applications, and fulfill their diverse resource demands and performance objectives with high scalability and flexibility. One challenge in this hybrid data center is that the dramatic fluctuation of batch and gratis workload may impact performance of production applications, cause task failure, decrease efficiency, and waste computing resources. One way to tackle the challenge is to reduce resource allocation to prevent host overload by delay scheduling tasks if resources are predicted in short. In this paper, we propose hierarchical prediction method for hybrid workload. We use last-state based ARMA model to predict stationary process of production workload, and use feedback based online AR model to predict the vibrated workload of batch and gratis tasks. Evaluation shows that the hierarchical prediction based task scheduling can reduce host overload by more than 85 percent, reduce tasks evicted and killed by more than 60 percent, and reduce 40 percent of average task scheduling delay.**

Keywords*—*hybrid data center; task scheduling; hierarchical prediction; last-state based ARMA model; feedback based online AR model

I. Introduction

Data center has gained significant popularity as a cost-effective platform. But in traditional data center, a tremendous amount of resource is over-provisioned to production applications for accommodating fluctuating workloads and peak demands as they have high SLA requirement and are of most importance for the enterprise. Although hosts in data center usually are 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 requirement of latency and SLAs and leveraging the over-provisioned resource effectively. The data center then can fulfill diverse resource demands and performance objectives of hybrid tasks with high scalability and flexibility.

In such hybrid data center, tasks are typically divided into three categories according to their priorities. Original tasks and jobs hosted on the server are production workload and have highest priorities. They are latency-sensitive and should not be killed due to over-allocation of machine resources. Batch tasks such as MapReduce and Dyrad, has middle priorities. They are insensitive to latency and can be killed when the host is overload. Gratis tasks have lowest priorities, and can be evicted by tasks with higher priorities or killed once the host is detected to be overload.

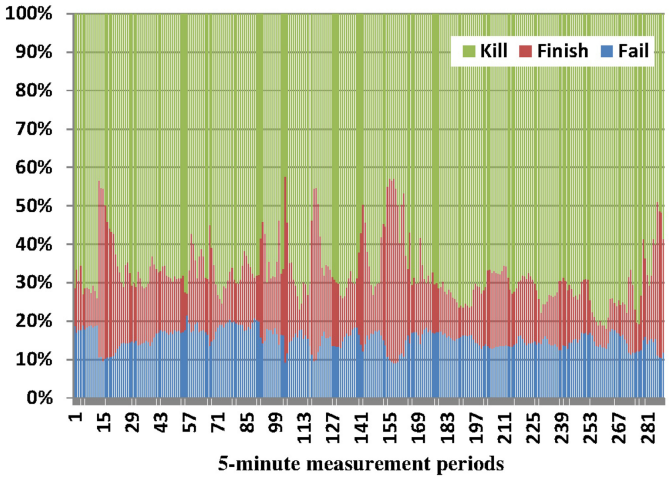


Fig. 1. Proportions of temporal CPU usage measured every 5 minutes [4].

In traditional batch task scheduling, when a new task comes into the cluster, it will be scheduled to a host with enough space or evict tasks with lower priorities if no host has enough for it. In the hybrid data center, this mechanism will bring two challenges. First challenge happens when the production application needs more resource but the spare capacity is fully allocated to batch and gratis tasks. In this situation, the performance of production application will be affected before killing some tasks. Second challenge happens when a task with higher priority come to the cluster shortly after a gratis scheduled and use up all space. The task with higher priorities will evict some gratis and be scheduled. These evicted and killed tasks will waste CPU cycle, and the scheduling, evicting, killing process also brings more overhead. As shown in Figure 1 of Google cluster trace data [4], about 60% of CPU cycles are used for tasks that were killed, and 10-20% of CPU cycles are spent on tasks that eventually fail, leaving only less than 20% of CPU cycles for tasks that complete normally. We need prediction algorithm to detect the challenge and strategy to tackle the problem.

Production and batch and gratis tasks have quite different workload features. As shown in Figure 2, production CPU usage is usually stable with daily peak-to-mean ratio approximately 1.3 and also has the most noticeable diurnal and weekly patterns. Production applications have highest priority, and cannot be evicted or killed when the host is overload. So the production workload pattern will not be affected by task scheduler. An accurate prediction model trained offline will fit requirement of production workload prediction. Usage of batch

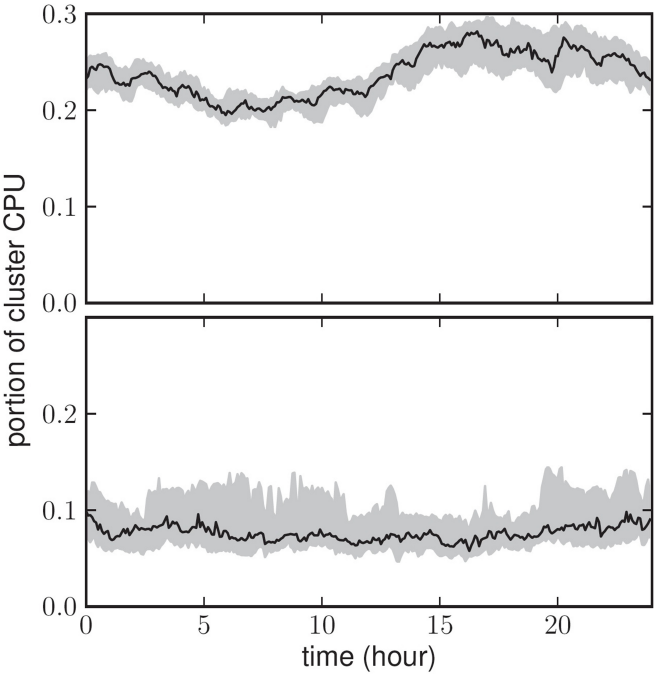


Fig. 2. Normal production (top) and lower (bottom) priority CPU usage by hour of day of Google cluster [5].

and gratis is much more irregular. It is not dominated by any daily or weekly pattern but has a much larger peak-to-mean ratio than the productive tasks. When host resource is in short, task scheduler will kill executing tasks or delay scheduling new tasks. It will change workload pattern, making prediction model using offline training inaccurate.

In the hybrid data center, host workload is calculated by real resource usage of production application plus total percentage of resource allocated to all the other tasks on the host. In this context, we use the term utilization to refer to production application utilization as well as percentage of resource allocated to the batch and gratis for simplicity. In this paper, we present a hierarchical prediction method according to different features of hybrid tasks, and make different task scheduling strategy according to the prediction result. We use Auto-Regressive and Moving Average (ARMA) model [6], [7] to predict stationary process of production workload with last-state to correct the result, and feedback based online Auto-Regressive (AR) [6], [7] model to predict the time-varying workload of batch and gratis tasks. When scheduling a batch task, enough space should be reserved for maximum workload of production applications. When scheduling a gratis task, it needs make sure that it will not use up all available resources and cause serious eviction when the next task come in. We then analyze he hieratical prediction based task scheduling by simulating real workload traces from Googles compute clusters. We show our proposed solution is capable of significantly reduce task killed and evicted complying with SLA in terms of task scheduling delay.

II. Related work

In the research literature on task scheduling in data center, many researches focus on improving batch task performance by improving data locality of MapReduce tasks [8]–[11]. Some works research fairness between different users [12]. R. C. Chiang and H. H. Huang [13] and X. Bu et al. [14] find out that cohosted applications may cause interference and suffer from performance degradation. They propose linear, quadratic and exponential models to predict task performance. M. Zaharia et al. [15] focus on data center with hybrid hardware and software while Q. Zhang et al. [16] focus on production data center with production tasks with hybrid resource demand, durations, priorities and performance objectives. B. Sharma et al. [17] propose a hybrid data center with both interactive and batch workloads, as well as both virtual and native machines in the data center. He utilizes available unused resources by consolidating batch jobs with over provisioned foreground applications to improve application performance and energy efficiency. He just uses interference prevention system to monitor interference, and kill tasks after interference happens. The afterward correction only works after problem happen, and cannot effectively prevent serious performance degradation. So we need a beforehand estimation method to detect potential risks and take action before performance degradation happens.

Literature researches on host load prediction usually aim at providing benchmark for virtual machine migration, server consolidation and energy management. M. A. Farahat et al. [18] used curve fitting prediction (CFP) technique combined with genetic algorithms (GAs) to obtain the optimum parameters of Gaussian prediction model.It provides very accurate hourly load forecast, but the overhead of the prediction is too high for real time task scheduling. A. Khan et al. [19] grouped VM first, and then designed a model to capture the CPU workload of different groups by leveraging the Hidden Markov Model (HMM). Q. Yang et al. [20] combines the Phase Space Reconstruction (PSR) method and the Group Method of Data Handling (GMDH) based on 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. D. Yang et al. [21] proposed a new multi-step-ahead prediction approach for CPU load that is more accurate than repeating the one-step-ahead prediction approach in three steps, namely find a fit function for the change range sequence, predict the change pattern, compose change range and change patter prediction. S. Di et al. [22], [23] use Bayes model to change prediction process into a classification problem, identify novel predictive features of host load, and predict the mean load over a long-term time interval. Problem of the Bayes model is 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. The above method can either gain a good accuracy or predict more than one step, but parameters or pattern of the model are all static and fixed in the prediction process. In our task scheduling process, we decide scheduling the new task whether or not according to the predicted workload. If the new task is queued, the workload trend is changed manually, making old parameters not fit at all. So we need a feedback based model to get parameters dynamically to adjust to the ever changing workload pattern. In the meantime, productive workload should be predicted separately to gain accuracy, and ARMA model is simple and accurate enough for it.

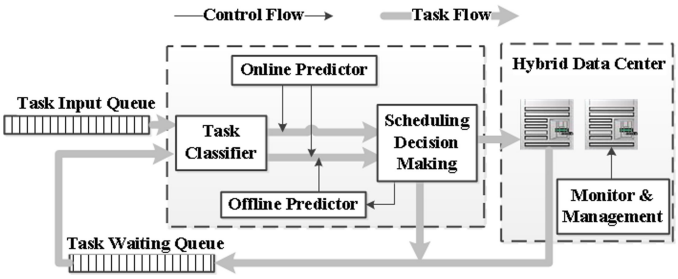


Fig. 3. Hierarchical Prediction based Task Scheduling Architecture.

III. System architecture

Our proposed system architecture is depicted in Figure 3. It consists of the following components:

- Task classifier identifies the category of input tasks and import different predictor for the task. For a batch task, only offline predictor is used, while for a gratis, both offline and online predictor are used.
- Offline predictor uses last-state based ARMA model with static parameters that is studies from training data center offline. The last recorded load value will be used to correct ARMA result. The offline predictor predicts the maximum resource utilization of production workload in the next prediction window. Then it suggests maximum amount resource capacity that can be safely allocated to the new task.
- Online predictor uses AR model with parameters dynamically updated to predict the maximum resource utilization of batch and gratis workload in the next prediction window. The suggested maximum resource amount that can be safely allocated to the gratis task should consider production workload plus batch and gratis workload.
- Scheduling decision making module decides scheduling operation for the new task according to the task priority and suggested available capacity provided by predictors. When the suggested available capacity is not enough for the task, a batch task will evict gratis and be scheduled, while a gratis will be queued and rescheduled until some tasks complete and release resources.
- Monitor & management module is a background program running periodically to check total utilization of each host. Once a host is overload, the module will kill some tasks to release resources for production applications.

IV. Hierarchical workload prediction

Workload of a host is a continuous function of time. We divide continuous time into discrete time slots. Maximum workload in the time slot is the value of time series. Then we get the time series $\{X_n\}$ of the host.

In hybrid data center, time series of production workload is stationary process. We use ARMA model to predict the production workload. If predicted value is lower than real one, more resources will be allocated and cause host overload. This will impact performance and SLAs of production applications. So we import last recorded state to correct the prediction. When the predicted value is lower than last recorded value, last recorded value is used as the prediction result.

Time series of batch and gratis workload shows evidence of non-stationarity, an initial differencing step can be applied to remove the non-stationarity and change the time series into stationary process. When the predicted workload is too high for the new gratis, delay scheduling will change the time series pattern. ARMA model will not work well as it relies heavily on prediction error of time n-1, which will be too large in the continuously changing pattern. So we use AR prediction model on the differenced time series with parameters updated dynamically based on feedback to meet time-varying workload series.

A. ARMA Prediction

Given that a stationary time series $\{X_n\}$ satisfies ARMA(p, q) model and the previous $n-1$ values $\{X_{n-1}\}$ are known, the expected value X_n at time n is formulated as follows:

$$\begin{aligned} &X_n - \varphi_1 X_{n-1} - \cdots - \varphi_p X_{n-p} \\ =&\varepsilon_n - \theta_1 \varepsilon_{n-1} - \cdots - \theta_q \varepsilon_{n-q}, \end{aligned} \tag{1}$$

where $\varphi_i,\ i = 1, 2, \cdots, p$ and $\theta_j,\ j = 1, 2, \cdots, q$ are parameters estimated from the previous $n-1$ values $\{X_{n-1}\}$. ε_k, $k = 1, 2, \cdots, q$ are error terms which are generally assumed to be independent identically distributed white Gaussian noise, namely,

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

It is generally considered good practice to find the smallest values of p and q which provide an acceptable fit to the data. Finding appropriate values of p and q in the ARMA(p,q) model can be facilitated using AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) [6].

ARMA models in general can, after choosing p and q, be fitted by least squares regression to find the values of the parameters which minimize the error term. For the training set of the time series satisfying ARMA(p,q) model, the vector of the parameters is donated as $\bar{\boldsymbol{\varphi}} = (\varphi_1, \cdots, \varphi_p, \theta_1, \cdots, \theta_q)^{\mathrm{T}}$. Estimated value of ε_n is donated as $\hat{\varepsilon}_n$. It is calculated recursively as follows:

$$\hat{\varepsilon}_n = \begin{cases} 0, & n \leq p \\ x_n - \sum_{i=1}^{p} \varphi_i x_{n-i} + \sum_{i=1}^{q} \theta_i \hat{\varepsilon}_{n-i}, & n = p+1, \cdots, N. \end{cases} \tag{3}$$

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

$$S(\bar{\boldsymbol{\varphi}}) = \sum_{n=p+1}^{N} \hat{\varepsilon}_n^2. \tag{4}$$

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

B. Feedback Based Online AR Prediction

ARMA(p,q) model consists of an AR(p) model and a MA(q) (MA means Moving Average) model, where AR(p) model is in the form as:

$$X_n = \varphi_1 X_{n-1} + \varphi_2 X_{n-2} + \cdots + \varphi_p X_{n-p} + \varepsilon_n, \quad (5)$$

meaning that X_n is calculated by the linear combination of previous values. And MA(q) is in the form as:

$$X_n = \varepsilon_n - \theta_1 \varepsilon_{n-1} - \theta_2 \varepsilon_{n-2} - \cdots - \theta_q \varepsilon_{n-q}, \quad (6)$$

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

We use prediction values to determine task scheduling operation. That will change workload pattern externally, which means the error term ε_k, $k = 1, 2, \cdots, q$ in MA(q) will be larger than it should be, making ARMA model inaccurate. In the meantime, fixed parameters do not satisfy the continuously changing pattern of time series. So we calculate parameters online dynamically based on the real workload feedback. Each time the workload is changed, we recalculate the parameters.

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 [6]. We use Yule-Walker equations shown as follows:

$$\begin{cases} \rho_1 = \varphi_1 + \varphi_2 \rho_1 + \cdots + \varphi_p \rho_{p-1} \\ \rho_2 = \varphi_1 \rho_1 + \varphi_2 + \cdots + \varphi_p \rho_{p-2} \\ \quad \vdots \\ \rho_p = \varphi_1 \rho_{p-1} + \varphi_2 \rho_{p-2} + \cdots + \varphi_p, \end{cases} \quad (7)$$

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

$$\rho_k = \rho_{-k} = \frac{\gamma_k}{\gamma_0}, \quad k \geq 0, \quad (8)$$

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

$$\gamma_k = \gamma_{-k} = E[(X_n - \mu)(X_{n-k} - \mu)], \quad k \geq 0. \quad (9)$$

Especially,

$$\gamma_0 = E[X_n X_n] - E[X_n]E[X_n] = \sigma^2 - \mu^2. \quad (10)$$

We use AR(2) model in this paper for simplicity and accuracy. From the Yule-Walker equation, we have:

$$\begin{aligned} \varphi_1 &= \frac{\rho_1(1-\rho_2)}{1-\rho_1^2} = \frac{r_0 r_1 - r_1 r_2}{r_0^2 - r_1^2}, \\ \varphi_2 &= \frac{\rho_2 - \rho_1^2}{1-\rho_1^2} = \frac{r_0 r_2 - r_1^2}{r_0^2 - r_1^2}. \end{aligned} \quad (11)$$

Time series of batch and gratis workload show evidence of non-stationarity. Thus, an initial differencing step is applied to remove the non-stationarity. The first order differenced time series $\{\Delta X_n\}$ of AR(p) model is in the form as:

$$\Delta X_n = \varphi_1 \Delta X_{n-1} + \varphi_2 \Delta X_{n-2} + \cdots + \varphi_p \Delta X_{n-p} + \varepsilon_n. \quad (12)$$

For the first order differenced AR(2) model,

$$\begin{aligned} &X_n - X_{n-1} \\ =&\varphi_1(X_{n-1} - X_{n-2}) + \varphi_2(X_{n-2} - X_{n-3}) + \varepsilon_n. \end{aligned} \quad (13)$$

So we get the following equation for the batch and gratis workload prediction:

$$X_n = (1+\varphi_1)X_{n-1} + (\varphi_2 - \varphi_1)X_{n-2} - \varphi_2 X_{n-3} + \varepsilon_n. \quad (14)$$

Each time when we get 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.

Algorithm 1
Pseudo code for task schedule in hybrid data center

```
Input: T: queue of incoming tasks
       H: set of hosts in the data center
 1:  for each task t_i in T = {t_1, t_2, ..., t_n} do
 2:    if t_i is batch task do
 3:      for each host h_j in H
 4:        Calculate available resource using last-state
           based ARMA model
 5:      end for
 6:      Choose the h_j with maximum available
         resources
 7:      if maximum available resources > task
           request do
 8:        Schedule t_i to h_j and update parameters
           of AR(p) model
 9:      end if
10:      else do
11:        Choose a host h_j randomly
12:        Evict gratis tasks in h_j randomly until
           available resources > task request
13:        Schedule t_i to h_j and update parameters
           of AR(p) model
14:      end else
15:    end if
16:    if t_i is gratis task do
17:      for each host h_j in H
18:        Calculate available resources using last-state
           based ARMA model and AR(p) model
19:      end for
20:      Choose the h_j with maximum available
         resources
21:      if maximum available resources > task
           request do
22:        Schedule t_i to h_j and update parameters
           of AR(p) model
23:      end if
24:      else put t_i into waiting queue
25:    end if
```

V. Task Schedule and Resource Management Method Design

The hierarchical prediction based task scheduling algorithm is shown in Algorithm 1. When a batch task comes, we calculate the available resource using last-state based ARMA model (Line 4) and choose the host with most available resources (Line 6). For simplicity, if there is not enough space, we randomly evict gratis tasks until the batch task can be scheduled with enough available resources (Line 11-13). When a gratis comes, we calculate the available resource using last-state based ARMA model plus AR(p) model (Line 18) and choose the host with most available resources (Line 20). If

TABLE I. EVALUATION ENVIRONMENT.

MIPS means million of instructions per second

Machine ID	CPU rate (MIPS)	CPU cores	Memory capacity
0	2000	1	1
1	1000	0.5	0.2393
2	1500	0.5	0.4995
3	1000	0.5	0.4995
4	1500	0.5	0.4995
5	1000	1	1

TABLE II. PRIORITIES OF THE TASKS.

Priority	Number of tasks	Task categories
0	1754	Gratis tasks
1	475	Batch tasks
2	58	Batch tasks

there is not enough space, the task is put into waiting queue (Line 24) and rescheduled when some tasks finish and release enough resources. Each time of scheduling a new task, online recalculation of AR(p) parameters is invoked (Line 8, 13, 22). There is also another monitor process running periodically to detect host overload. Once the monitor detect overload, it kills gratis tasks randomly until overload is eliminated.

VI. EXPERIMENTAL EVALUATION

In this section, we conduct trace-driven simulations to realistically evaluate the performance improvement of task by using hierarchical prediction in task scheduling. Our simulations are based on real-world workload traces Google cluster [24] and Cloudsim [25] toolkit.

A. Simulation Setup

We simulate the task constraints, machine features and task submission time of Google cluster. We choose six machines with different CPU size, CPU rate, memory size and disk to simulate a data center heterogeneous capacity. Table I lists the configuration. Each machine is allocated 10kMB image size, and the virtual machine management version is Xen. CPU and memory capacity is normalized to 1 consistent with Google cluster.

One day record on the six machines is chosen to represent the task flow. Tasks are divided into three categories according to priorities [5],namely production, with priorities of 9-11, batch ,with priorities 2-8, and gratis, with priorities 0-1. Production tasks are deleted from input task flow. We simulate the original production workload on the host with real CPU usage of production tasks on the machines. Input set of tasks only consists of batch and gratis tasks. In our experiment, there are 2017 tasks submitted in one day, and priorities of the tasks is shown in Table II.

Slot representing length of prediction widow is determined by task completion time. It should cover most task completion time to gain best accuracy. Task completion time is calculated by task processing time plus file transmission time shown as follows [25]:

$$T_{complet} = T_{task} + T_{file} = \frac{L_{task}}{R_{CPU}} + \frac{L_{file}}{bandwidth}. \quad (15)$$

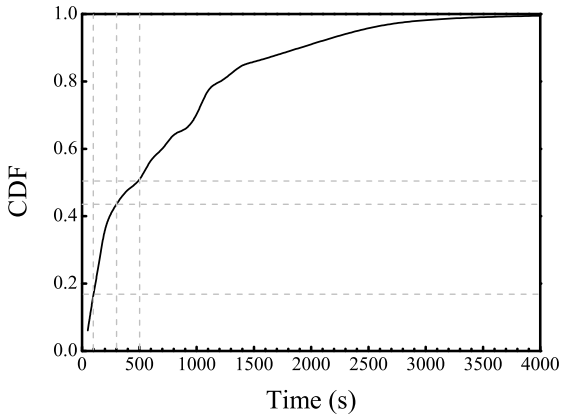


Fig. 4. CDF of task completion time.

Task processing time is calculated by task length divided by CPU processing rate. In our simulation, CPU rate is preset, and file is not taken into consideration. So task completion time is fixed during execution. Figure 4 shows CDF (cumulative distribution function) of task completion time. Half tasks can finish in 500s and more than 40 percent of tasks can finish in 300s. So it is reasonable to set slot=300s or slot=500s.

B. Prediction Performance

Our first experiment evaluates the performance of the ARMA prediction for production workload, and shows improvement of accuracy by importing last recorded state to correct ARMA. We use IBM SPSS Statistics [26] toolkit to identify ARMA(1,1) model and get parameters from offline training.

If predicted value is higher than real one, resource allocated to new tasks will be lower. It will just waste a few resources, and the resource will be allocated in the next scheduling time. But if predicted value is lower than real one, more resources will be allocated and cause host overload. This will impact production applications performance and kill more tasks.

We use the mean squared error (MSE) to evaluate accuracy of prediction. MSE is calculated as follows:

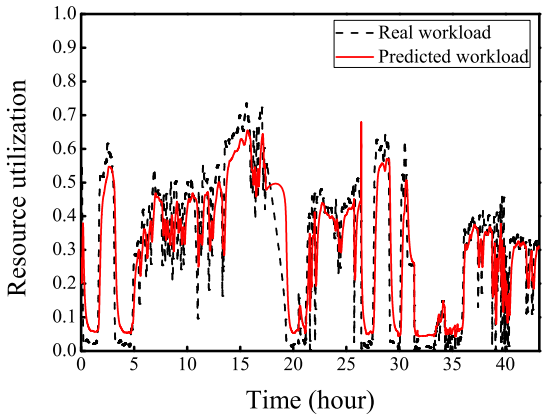
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{X}_i)^2, \quad (16)$$

where X_i is the real workload and $\hat{X}_i$ is the predicted workload.

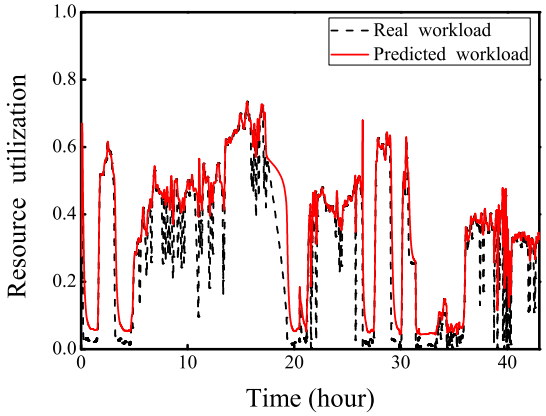
Figure 5 shows the prediction of the production workload compared to the real workload. It shows that points of under estimation is greatly reduced by importing last recorded state to the ARMA(1,1) model. MSE also decreases from about 0.007126 to about 0.003699.

C. Task Scheduling Performance

In this section, we evaluate effects of hierarchical prediction based task scheduling. We set the slot 300 seconds to match Googles Cluster measurements frequency.



(a) Prediction with ARMA(1,1) model, MSE ≈ 0.007126



(b) Prediction with last-state based ARMA(1,1) model, MSE ≈ 0.003699

Fig. 5. Prediction of production workload.

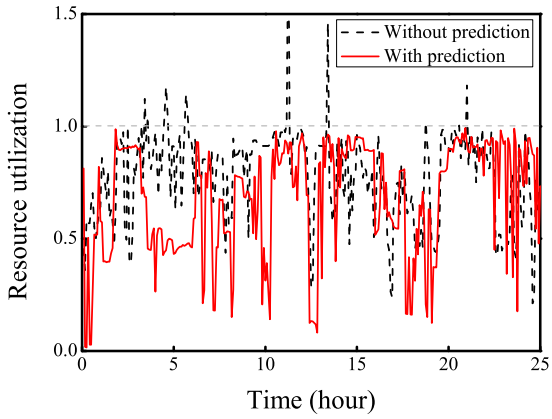


Fig. 6. Host workload change using hierarchical prediction.

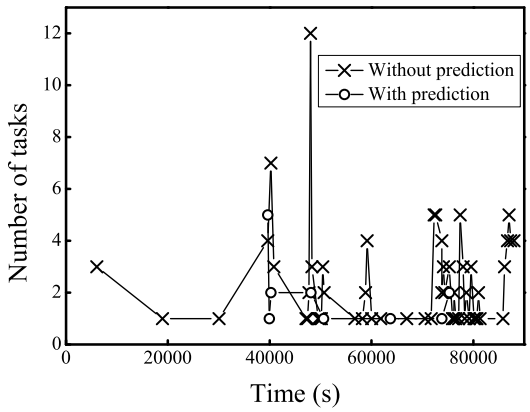


Fig. 7. Number of killed tasks reduced by using hierarchical prediction.

In our experiment, host workload is calculated by real resource utilization of production application plus total percentage of resources allocated to all the other tasks on the host. A background program monitors each host periodically and once the utilization is over 100%, some tasks with low priorities is killed. The reason why workload can exceed 100% is because resource allocated to batch and gratis is always larger than real usage. The monitor method is based on post feedback. We record the maximum workload of each slot. Figure 6 shows a host workload changes after the hierarchical prediction based task scheduling. Peak of workload is diminished greatly, and times of overload decrease greatly from 116 to 14, which is more than 85% improvement. This is because hierarchical prediction based task scheduling provides a regulatory operation before the peak comes.

As host overload is greatly diminished, the operation of killing tasks to release resources to production applications is of course not used. Figure 7 shows that number of killed tasks reduced after using hierarchical prediction. There are 132 tasks killed before, and only 15 after using prediction, almost reducing 88% killing operation. It saves a great deal of CPU cycles wasted by the killed tasks, saves energy and time used for scheduling and killing operations, and most importantly maintains performance and SLAs of production applications.

Figure 8 shows that number of evicted tasks reduced after using hierarchical prediction. There are 404 tasks evicted before, and 128 after using prediction, almost reducing 65% eviction operation. That saves a great deal of CPU cycles wasted by the execution of evicted tasks and saves energy and time used for scheduling and killing operations.

When the predicted available capacity is not enough for a new task, the task is put into the waiting queue. This may bring extra task scheduling delay. But killing or eviction of a task also brings scheduling delay. Figure 9 shows CDF of task scheduling delay. It shows that average scheduling delay decreases by 40% from 573s to 333s. There are 1826 tasks scheduled without delay before hierarchical prediction and the number increases to 1995 after hierarchical prediction. It is because when a task is put into queue, it leaves spare resources to the latter tasks that have fewer resource requests can be scheduled immediately. The longest delay also decreases form more than 50000s to less than 30000s. This improvement may be caused by stoping scheduling and killing or evicting the same task repeatedly.

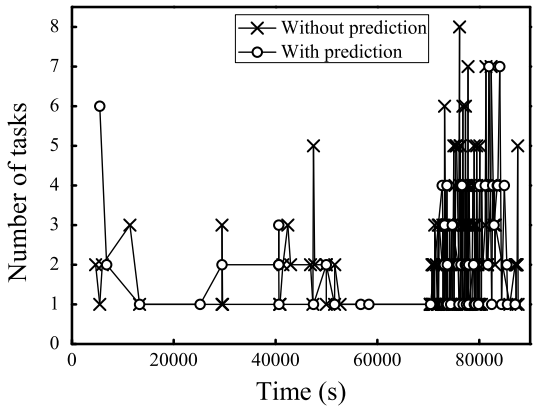


Fig. 8. Number of evicted tasks reduced by using hierarchical prediction.

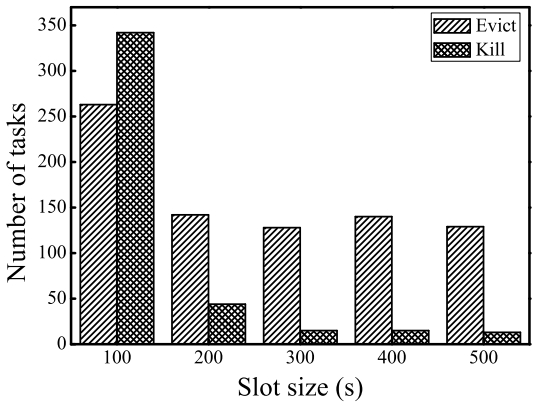


Fig. 10. Number of tasks evicted and killed of different slot sizes.

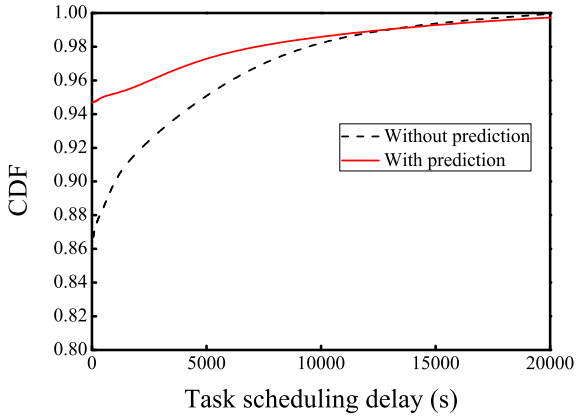


Fig. 9. CDF of task scheduling delay before and after hierarchical prediction.

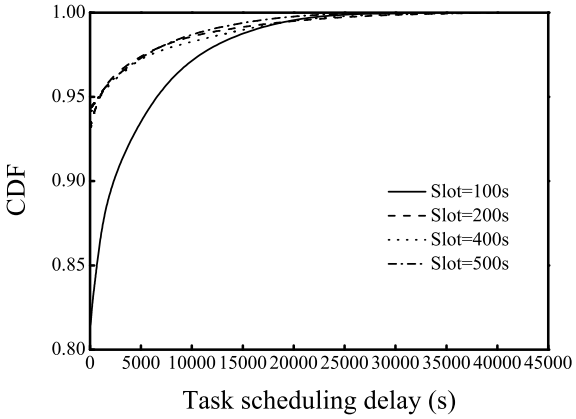


Fig. 11. CDF of task scheduling delay of different slot sizes.

D. Comparison of Different Slot Sizes

We divide continuous time into discrete slots. Each slot is a prediction widow. It is determined by task completion time and will affect prediction accuracy. If it is too small to cover most task completion time, the predicted available capacity will not have enough effect on schedule. Figure 10 shows number of tasks evicted and killed of different slot sizes. When slot is set 100s, it performs even worse than task scheduling performance without prediction. It performs well when slot size is larger than 200s. Performance does not vary greatly of slot size larger than 200s, as task completion time is uniformly distributed from 200s to 500s. Figure 11 shows CDF of task scheduling delay of different slot sizes. Table 3 shows detailed task scheduling delay features. It shows that the slot size of 300s performs best in task scheduling delay.

TABLE III. TASK SCHEDULING DELAY FEATURES OF DIFFERENT SLOT SIZES.

SLOT size(s)	Number of tasks scheduled without delay	Average scheduling delay (s)	Max scheduling delay (s)
100	1715	777.07	30777
200	1964	399	34588
300	1995	333	24619
400	1984	406	33473
500	1988	334	32216

VII. CONCLUSION

Over-provision to production applications in traditional data center causes a waste of resources. Cloud computing can help data center consolidate batch and gratis tasks with production applications effectively. In this paper, we tackle challenges of hybrid data center by hierarchically predict workload before scheduling a new task and make different scheduling decisions according to the task priorities. We use last-state based ARMA model to predict stationary process of production workload and use a feedback based online AR prediction model to meet the dramatic fluctuations of batch and gratis workload. The amount of resources provided to tasks is limited to avoid causing overload in the next slot instead of all the available resources. Gratis tasks will be delayed if it is predicted to cause overload in the next slot. Evaluation shows that our hierarchical prediction based task scheduling method can reduce host overload by more than 90 percent, reduce tasks eviction and killing by more than 60 percent, reduce

average task scheduling delay by 40 percent, and increase number of tasks scheduled immediately by 5 percent. It means that our hierarchical prediction based scheduling can maintain performance of production applications effectively and save computing resources greatly.

As part of the future work, we plan to investigate the interference between hybrid tasks and find the upper limit of available resources for new tasks while avoiding interference. Also, we plan to research on the variable length of slot according to the completion time of the task to be scheduled instead of fixed slot used in this paper, while preserve accuracy and minimize scheduling overhead.

ACKNOWLEDGMENT

This work is supported by Ministry of Education - China Mobile Research Fund (2012); the National Key project of Scientific and Technical Supporting Programs of China (Grant No. 2012BAH01F02, 2013BAH10F01, 2013BAH07F02).

REFERENCES

[1] B. Guenter, N. Jain, and C. Williams, "Managing Cost, Performance, and Reliability Tradeoffs for Energy-Aware Server Provisioning," IEEE INFOCOM, 2011.

[2] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," Concurrency and Computation: Practice and Experience, vol. 24, pp. 1397C1420, 2012.

[3] A. A. Bhattacharya, D. Culler, E. Friedman, A. Ghodsi, S. Shenker, and I. Stoica, "Hierarchical Scheduling for Diverse DatacenterWorkloads," The IEEE International System-on-Chip Conference, pp. 1-15, 2013.

[4] Z. Liu and S. Cho, "Characterizing Machines and Workloads on a Google Cluster," 41st International Conference on Parallel Processing Workshops, pp. 397-403, 2012.

[5] C. Reiss and A. Tumanov, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis," The IEEE International System-on-Chip Conference, 2012.

[6] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel, Time Series Analysis: Forecasting and Control, 4th ed., China Machine Press, pp. 56-81, 2011.

[7] L. Hua and H. Qiying, Forecasting and Decision Making, China Machine Press, pp. 131-168, 2012.

[8] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map Task Scheduling in MapReduce with Data Locality: Throughput and Heavy-Traffic Optimality," INFOCOM, pp. 1609-1617, 2013.

[9] Z. Guo, G. Fox, and M. Zhou, "Investigation of Data Locality in MapReduce," CCGrid, 2012.

[10] B. Palanisamy, A. Singh, L. Liu, and B. Jain, "Purlieus: Locality-aware Resource Allocation for MapReduce in a Cloud," In Proceedings of the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis, SC,, 2011.

[11] M. Li, D. Subhravet, A. R. Butt, A. Khasymski, and P. Sarkar, "CAM: A Topology Aware Minimum Cost Flow Based Resource Manager for MapReduce Applications in the Cloud," HPDC, pp. 211-222, 2012.

[12] M. Zaharia, D. Borthakur, and J. S. Sarma, "Delay Scheduling: A Simple Technique for Achieving Locality and Fairness in Cluster Scheduling," EuroSys, 2010.

[13] R. C. Chiang and H. H. Huang, "TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments," In Proceedings of the IEEE/ACM Conference on High Performance Computing Networking, Storage and Analysis, SC,, 2011.

[14] X. Bu, J. Rao, and C.-Z. Xu, "Interference and Locality-Aware Task Scheduling for MapReduce Applications in Virtual Clusters," HPDC, pp. 227-238, 2013.

[15] M. Zaharia, A. Konwinski, A. D. Joseph, and R. K. I. Stoica, "Improving MapReduce Performance in Heterogeneous Environments," Technical Report No. UCB/EECS-2008-99, 2008 [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-99.html

[16] Q. Zhang, M. F. Zhani, R. Boutaba, and J. L. Hellerstein, "HARMONY: Dynamic Heterogeneity Aware Resource Provisioning in the Cloud," IEEE 33rd International Conference on Distributed Computing Systems, pp. 510-519, 2013.

[17] B. Sharma, T. Wood, and C. R. Das, "HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers," IEEE 33rd International Conference on Distributed Computing Systems, pp. 102-111, 2013.

[18] M. A. Farahat and M. Talaat, "Short-Term Load Forecasting Using Curve Fitting Prediction Optimized by Genetic Algorithms," International Journal of Enegry Engineering, vol. 2, pp. 23-38, 2012.

[19] A. Khan, X. Yan, T. Shu, and A. Nikos, "Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach," 13th IEEE/IFIP Network Operations and Management Symposium, pp. 1287-1294, 2012.

[20] Q. Yang, C. Peng, Y. Yu, H. Zhao, Y. Zhou, Z. Wang, and S. Du, "Host Load Prediction Based on PSR and EA-GMDH for Cloud Computing System," IEEE Third International Conference on Cloud and Green Computing, pp. 9-15, 2013.

[21] D. Yang, J. Cao, C. Yu, and J. Xiao, "A Multi-step-ahead CPU Load Prediction Approach in Distributed System," Second International Conference on Cloud and Green Computing, pp. 206-213, 2012.

[22] S. Di, D. Kondo, and W. Cirne, "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.

[23] S. Di, D. Kondo, and W. Cirne, "Google Hostload Prediction based on Bayesian Model with Optimized Feature Combincation," Journal of Parallel and Distributed Computing, pp. 1820-1832, 2014.

[24] "google cluster data," 2011. [Online]. Available: http://code.google.com/p/googleclusterdata/wiki/ClusterData2011_1

[25] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. e. A. F. D. Rose, and R. Buyya, "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," SoftwareCPractice and Experience, vol. 41, pp. 23C50, 2011.

[26] "IBM SPSS Statistics," [Online]. Available: http://www.spss.co.in/