Cloud Analytics for Capacity Planning and Instant VM Provisioning

Yexi Jiang, Chang-Shing Perng, Tao Li, and Rong N. Chang, *Senior Member, IEEE*

***Abstract*—The popularity of cloud service spurs the increasing demands of virtual resources to the service vendors. Along with the promising business opportunities, it also brings new technique challenges such as effective capacity planning and instant cloud resource provisioning.**

In this paper, we describe our research efforts on improving the service quality for the capacity planning and instant cloud resource provisioning problem. We first formulate both of the two problems as a generic cost-sensitive prediction problem. Then, considering the highly dynamic environment of cloud, we propose an asymmetric and heterogeneous measure to quantify the prediction error. Finally, we design an ensemble prediction mechanism by combining the prediction power of a set of prediction techniques based on the proposed measure.

To evaluate the effectiveness of our proposed solution, we design and implement an integrated prototype system to help improve the service quality of the cloud. Our system considers many practical situations of the cloud system, and is able to dynamically adapt to the changing environment. A series of experiments on the IBM Smart Cloud Enterprise (SCE) trace data demonstrate that our method can significantly improve the service quality by reducing the resource provisioning time while maintaining a low cloud overhead.

***Index Terms*—Cloud computing, data mining, cloud analytics, capacity planning, instant provisioning.**

I. Introduction

Due to their flexibility, convenience and low cost, cloud services gradually become a ubiquitous choice for modern IT-solution of business activities. The paradigms such as IaaS, PaaS, and SaaS provide different styles of services to the cloud resource users with different flavors. Among these paradigms, IaaS is an elastic and economical choice for business IT support. It enables the cloud customers to dynamically request proper amount of virtual machines (VMs) based on their business requirements.

As cloud computing promises the pay-as-you-go flexible charging style, the resource demand becomes more volatile than that in traditional IT environments. Under such a circumstance, the aspects of *effective cloud capacity planning* and *instant on-demand VM provisioning* face new challenges. In principle, both of the tasks are about properly preparing the resources. More specifically, they both aim to ensure that the prepared resources in cloud is consistent with the real demand in the near future. However, due to the pay-as-you-go style of cloud services, the prediction of resource demands in cloud is a very challenging task [1].

A. Cloud Capacity Planning

The capacity planning in traditional services is straightforward. To satisfy the increasing demands, the service vendors simply upgrade the data centers by continuously scaling up the infrastructure. However, the capacity planing for cloud is not that trivial due to its elastic service style. The demand fluctuates and the available cloud resources are not always fully utilized.

When the capacity is overestimated, the extra prepared but unused physical resources are purely wasted. As the annual energy cost of the cloud data centers counts for billions of dollars [2] and it makes up for 23% of the total amortized costs of the cloud, the energy saving is an important aspect issue when managing the cloud. Moreover, the unused physical resources not only cause energy waste, but also result in more early purchase costs. As the price of the same equipment is always going down, it is always better to purchase equipment later than earlier. Furthermore, an overestimated capacity will bring extra associated cost like network, labor, and maintenance, all of which are proportional to the scale of the infrastructure [3].

On the other hand, the underestimation of the cloud capacity would cause resource shortage and revenue loss. For the cloud platform, hardware resources still require a long acquisition and deployment process. If the actual demand is higher than the existing capacity, the cloud has to postpone serving new customers, and thus lose the potential revenue. Once the shortage is severe, even existing services from existing customers would be affected, which defeats the promise that application in cloud can scaling-up whenever workload increases.

B. VM Provisioning

As for the issue of VM provisioning, the volatile style of cloud service makes the provisioning and de-provisioning of VM occur frequently. The state-of-art virtual machine (VM) provisioning technology is able to provision a VM in minutes. This time delay is affordable for normal services but is unacceptable for tasks that need to scale out during computation. To truly enable the on-the-fly scaling, new VM needs to be ready in seconds upon requests. However, from the infrastructure perspective, there is dim hope that new invented cloud-specific device can immediately and significantly reduce the request fulfillment time. The streaming VM technology [4] allows the user to preview the VM before it is entirely prepared, but the VM is still not instantly available until

Manuscript received April 11, 2012; revised March 1, 2013. The associate editor coordinating the review of this paper and approving for publication was R. Campbell.

Y. Jiang and T. Li are with the School of Computer Science, Florida International University, Miami, FL (e-mail: {yjian004, taoli}@cs.fiu.edu).

C.-S. Perng and R. N. Chang are with the IBM T.J Watson Research Center, Hawthorne, New York (e-mail: {perng, chang}@us.ibm.com).

Digital Object Identifier 10.1109/TNSM.2013.051913.120278

1932-4537/13/$31.00 © 2013 IEEE

enough proportion of it is ready. The rapid VM cloning techniques [5] enables a VM to be created in seconds. However, since the VM preparing involves a lot of procedures include VM creation, software installation, automatic patching, VM verification, security checking, and profiling, the acceleration of VM creation alone cannot significantly reduce the overall waiting time. From the business perspective, a simple yet straightforward solution to reduce the request fulfillment time is to ask all the customers to provide future VM demand schedule. However, this idea is impossible for many practical reasons: (1) The promise of cloud service is to provide the VM resources whenever they are needed. It is contradicting to ask the customers provide their plan well in advance. Also, the customers maybe unwilling to provide such schedules. (2) The customers themselves are hard to know when they need the resources, since the demand is associated with the actual business activity in the future. (3) The constituents of customers are always changing. New customers can join in and old customers can leave at any time. (4) Even if the customers provide their schedules. The actual schedules may change at any time.

Facing these technology limitations and business constraints, we believe that the only practical, effective and achievable solution to provide instant cloud is to *predict the demands and prepare the VMs in advance*. Similar to the capacity planning scenario, different types of VM mis-provisioning would also cause different consequences. If the customer sends a request for a certain type of VM and there is no prepared VM that matches this request, the cloud needs to provision the VM on-the-fly, which, as mentioned before, is time-consuming. In this situation, the Service Level Agreement (SLA) would be violated and the penalty is high (The penalty can be more than a thousand dollars per violation). On the other hand, if the prepared VM is not consumed by any customer, it is idled. This would not only make the prepared VM itself to be wasted, but also indirectly preempt the physical resources that could be allocated to other useful VMs. If the mis-provisioning is severe, although the workload of the cloud is high, its real utility is low.

C. Paper Contribution and Organization

In our prior works [6][7][8], we tackled the capacity planning and instant on-demand VM provisioning problem with separate models and solutions. In this paper, we propose an integrated solution to address these two problems within a unified framework. Generally, both problems can be first formulated as a generic time series prediction problem. Due to the pay-as-you-go style of cloud service, the prediction is a very challenging task. The unbalanced demand distribution, dynamic changing requests, and continuous customer turnover makes the prediction difficult. Our empirical studies show that applying traditional time series prediction techniques on these two problems cannot achieve acceptable performance. Moreover, traditional prediction techniques are unable to dynamically change the prediction strategy according to actual cloud environment. Based on aforementioned facts, we introduce a cost-sensitive model to quantify the heterogeneous cost caused by the mis-prediction. Using the proposed model, we design a domain-specific self-training prediction mechanism to solve the prediction problem.

Specifically, our contributions can be summarized as follows:

1) We formulate both the capacity planning and instant VM provisioning as the time series problem. And we introduce an asymmetric and heterogeneous measurement called *Cloud Prediction Cost* in the context of cloud service to evaluate the quality of our prediction results.
2) We design an integrated system that predicts the future demands by combining the prediction power of a set of state-of-art algorithms. Our system is able to properly predict the future capacity of the cloud based on ad-hoc setting. Also, this system is able to predict the future demand of each VM type. Especially, based on the temporal dynamics of the VM type importance, our system is able to dynamically take different pre-provisioning strategies.
3) We conduct a series of simulation experiments on the trace data of IBM Smart Cloud Enterprise (SCE) [9], a real world cloud environment, to demonstrate the effectiveness of our system.

The scope of this paper is confined to optimize the utility of existing resource based on the middle-term optimization strategy. The real-time cloud utility optimization is one of our future works.

The rest of this paper is organized as follows. We formulate the problem of capacity planning and VM provisioning as a time-series problem in Section II. Then we discuss our solution and the corresponding system framework in Section 3. In Section IV, we introduce the details about the prediction approach. An extensive evaluation results of our system are reported in Section V. Finally, we discuss the related works and summarize the paper in Section VI and Section VII, respectively.

II. Problem Description

In principle, both the problems of achieving effective cloud capacity planning and instant VM provisioning are about properly preparing the resources. More specifically, they both aim to ensure that the prepared resources in cloud is consistent with the real demand in the near future.

Without loss of generality, we use the number of *VM units* to quantify the amount of resources that is needed at a time slot. (We will discuss how we choose the proper time slot in later sections.) The *VM unit* is defined as the basic unit of virtual resource, which is associated with a set of physical resources such as CPU time, main memory, storage space, electricity etc. In real cloud systems, any virtual resource a customer can apply should be a multiple of the *VM unit*. For example, IBM's cloud product SCE defines one 64-bit *VM unit* as one 1.25 GHz virtual CPU with 2G main memory and 60G of storage space. The customer can request VMs with different multiples of the basic *VM unit*, such as *Copper*, *Bronze*, *Silver*, *Gold*, and *Platinum*. In the remaining of this paper, the amount of needed resources is quantified by the number of *VM units*.

Let $v^{(t)}$ be the future resource at a future time t in terms of number of *VM units*, and $\hat{v}^{(t)}$ be the corresponding prepared resources at time t, the goal of effective capacity planning and instant provisioning is to prepare the needed resources at each time slot t, such that the prediction error:

$$E = \sum_t f(v^{(t)}, \hat{v}^{(t)}) \tag{1}$$

is minimized. In Equation (1), $f(\cdot,\cdot)$ represents an arbitrary cost function that is used to quantify the prediction error. A good resource predictor should be able to reduce the error as much as possible.

Specifically, for the cloud capacity planning scenario, v semantically denotes the quantified amount of physical machines, associated available cooling systems, corresponding power supplies, and the number of system operators working for the cloud. As for the instant VM provisioning scenario, v represents the number of VM instances with a certain type that should be prepared before the customers actually send the requests. In this paper, we focus on the IaaS paradigm resource provisioning, so each VM is associated with a certain OS or middle-ware type, i.e. *Linux Red Hat Enterprise 5.5*, *Windows Server 2003*, etc.

III. The System Framework

According to the problem description mentioned above, a natural solution for both of the problems is to leverage the state-of-art time series prediction techniques to predict the future needed resources. However, based on our empirical study about the available SCE trace data, we found the following difficulties make the task non-trivial:

1) **The resource demands are highly unstable.** This is caused by two reasons: *the unstable customer constituents* and *the freestyle of resource acquisition/releasing*. In the presence of dynamic varying customer group, both the time series of the cloud capacity and the required VMs for each type may contain random factors that can mislead the prediction algorithms. Figure 1 shows the change of the customer number over time [1]. This figure implies that the number of customers is continually increasing. Therefore, even the old customer keep their request behavior, the overall request demands still change over time. This phenomena would cause the distributions of the resource demands to be unstable. Figure 2 illustrates the request history of three frequently requested customers. As is shown, these time series share no common property from each other.
2) **To enable instant provisioning, we need to predict the demand of each VM type separately, and then prepare them accordingly.** Since each type of VM has different characteristics in terms of demand amount, VM life-time, degrees of emergence, and physical resource requirements, their corresponding time series (See Figure 3) also show different temporal properties such as scale, degree of sharpness, length of cycle, degree of fluctuation, etc.

To properly address the above difficulties, we design and implement a flexible resource prediction system that enables the automatic cloud management. At the core of our system, we propose the ensemble time series predictor (we will discuss it in detail in Section IV) that combines the prediction power of a set of state-of-art prediction techniques. Besides combining individual predictors, we also consider the temporal correlations to ameliorate the prediction. Figure 4 illustrate the framework of our system. As is depicted, our system works as an associated system for the cloud. It mainly contains the following modules:

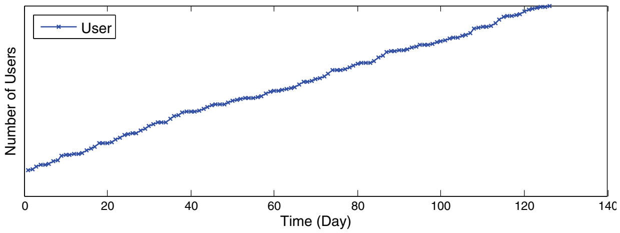


Fig. 1. The change of cloud service customer over time.

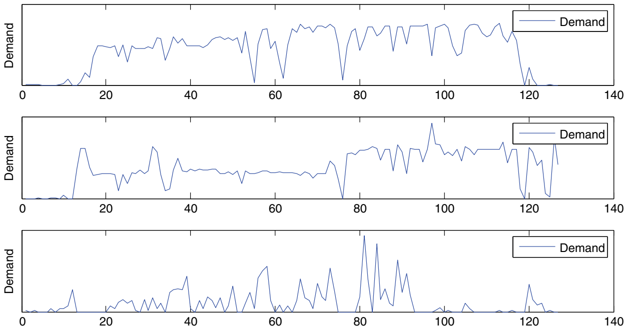


Fig. 2. Time series of resource demands group by customer. The three time series show the demand of three frequently requesting customers. The scales of all the time series are not normalized for ease of visualization.

1) *Data Preprocessing Module.* This module conducts the preprocessing works. It is responsible for: (1) filtering out the unnecessary information from the raw data; and (2) extracting the high-level characteristics of the filtered data and transform them into time series.
2) *Model Generation Module.* This module is responsible for building the models of separate predictors. It periodically selects and trains the most suitable models of each predictor based on the latest requests. Once the work is finished, the new parameters of the trained models are stored into a particular file and the demand prediction module is notified.
3) *Ensemble Prediction and Adjustment Module.* Once the individual prediction models are built, they are used to predict the future demands separately. Then an ensemble mechanism is used to obtain the final prediction results, and the results are sent to the configuration module. Finally, if necessary, the temporal correlations are considered to help adjust the prediction result. The details of this model will be introduced in Section IV.
4) *Configuration Module.* This module is responsible to configure the cloud based on the prediction results. It knows all the running status of the cloud, including the amount of available computing resources, the workload

[1] For confidential issue, we remove y-axis in all the time series.

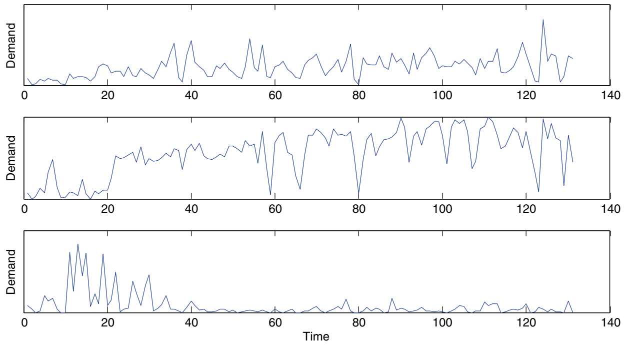


Fig. 3. Time series of resource demands group by VM types. The three time series show the demand of three frequently requested VM types. The scales of all the time series are not normalized for ease of visualization. For confidential issue, the value on y-axis are removed.

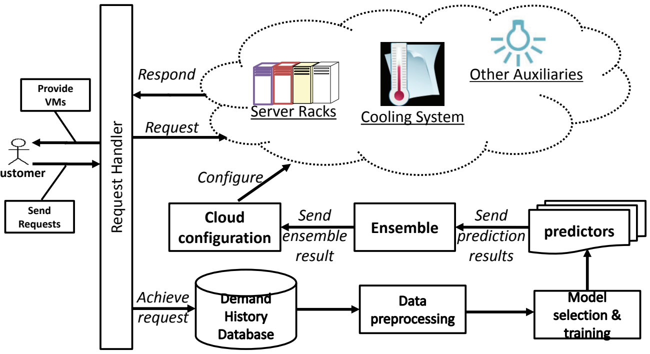


Fig. 4. The system framework.

of cooling system, and the power supply will dynamically adjust the cloud capacity based on the current status and the prediction results.

IV. Prediction Approach

In this section, we will first discuss the criteria as well as the cost function of how to quantify the prediction result error. Then we will introduce the ensemble prediction model based on the proposed cost function. Finally, we will discuss how to adopt this model to the problems of *capacity prediction* and *VM instant provisioning*.

A. Prediction Result Evaluation Criteria

Traditional regression cost measure such as *mean average error (MAE)*, *least square error (LSE)*, and *mean absolute percentage error (MAPE)* are all symmetric measures [10]. In cloud demand prediction scenario, the consequence of over-prediction and under-prediction are semantically different, and they result in heterogeneous costs. Therefore, a symmetric measures is not proper to model the asymmetric cost.

In cloud scenario, if the prepared resources are more than the actual demand, the customers would not be affected but the service vendor will suffer. This is because the idled and wasted resources are not billable. On the other hand, if the prepared resources are less than actual demand, the service quality of the customers would be affected and the SLA would be violated.

Considering the characteristics of the cloud, we propose a novel cost measure called *Cloud Prediction Cost (CPC)* to quantify the prediction result quality. *CPC* is an asymmetric and heterogeneous measure that models the prediction error as two different kinds of costs: the cost of SLA penalty and the cost of idled resources. Respectively, we use $P(v^{(t)}, \hat{v}^{(t)})$ and $R(v^{(t)}, \hat{v}^{(t)})$ to quantify their costs. The total costs can be represented as:

$$C = \beta P(v^{(t)}, \hat{v}^{(t)}) + (1 - \beta) R(v^{(t)}, \hat{v}^{(t)}). \tag{2}$$

Here β is used to tune the importance between two costs. Take the capacity prediction scenario for example, if the workload (the proportion of current capacity to the maximum capacity) of the cloud is low, the system operator can increase β, and thus the cost measure focusing more on the SLA penalty.

CPC is a generic cost measure that can be used for both capacity management and instant VM provisioning scenarios. Moreover, since different cloud has different objectives, the concrete quantification of the costs may vary accordingly. Basically, the R function and P function can be defined in any form that satisfies the following two properties:

1) *Non-negativity.* Both $P(v^{(t)}, \hat{v}^{(t)}) \geq 0$ and $R(v^{(t)}, \hat{v}^{(t)}) \geq 0$ should be hold for arbitrary non-negative $v^{(t)}$ and $\hat{v}^{(t)}$.
2) *Consistency.* If $v_1(t) - \hat{v}_1(t) \geq v_2(t) - \hat{v}_2(t)$, then $P(v_1(t), \hat{v}_1(t)) - P(v_2(t), \hat{v}_2(t))$ is either consistently positive or negative. Similarly, if $v_1(t) - \hat{v}_1(t) \geq v_2(t) - \hat{v}_2(t)$, then $R(v_1(t), \hat{v}_1(t)) - R(v_2(t) - \hat{v}_2(t))$ is either consistently positive or negative.

Predicting cloud resource is different from traditional time series prediction due to its asymmetric and heterogeneous characteristics, where the over-estimation and under-estimation of the resource would result in different consequences with different physical meaning. To measure the prediction cost intuitively, we define *the cost of SLA penalty* and *the cost of resource waste* as follows.

1) Cost of SLA penalty: The cost of SLA penalty is used to quantify the satisfaction of the customer. This cost is largely due to the under-estimation of the future resource requests. For simplicity, we use the request fulfillment time to quantify the SLA penalty. We denote the request fulfillment time of this situation as T_{avail}. When available resources are enough, the resources associated to the request can be immediately allocated to the requester. On the other hand, if there is not enough resources, the requester has to wait for T_{delay} until new resources are available and allocated. The time of wait is undecidable, but typically $T_{avail} \ll T_{delay}$ since this situation always involves garbage collection and resource reallocation.

In our system, we quantify the SLA penalty with P function and its definition can be seen as Equation 3. Based on the definition, the penalty is proportional to the degree of under-estimation. More sophisticated forms of the P function can also be used. For example, a common SLA typically specifies a penalty threshold for resource fulfillment time. The cost of a non-violated request in this case would have zero values for P function.

$$\begin{aligned} P(v^{(t)}, \hat{v}^{(t)}) = {} & min(v^{(t)}, \hat{v}^{(t)}) T_{avail} \\ & + max(0, v^{(t)} - \hat{v}^{(t)}) T_{delay}. \end{aligned} \tag{3}$$

TABLE I
TIME SERIES PREDICTION TECHNIQUES USED FOR ENSEMBLE

Method Name	Description
Moving Average	Naive Predictor
Auto Regression	Linear Regression
Artificial Neural Network	Non-linear Regression
Support Vector Machine	Linear Learner with Non-linear Kernel
Gene Expression Programming	Heuristic Algorithm

2) Cost of resource waste: This is the non-billable part on the service vendor side. It includes the server aging, electricity waste and labor cost etc. We use R_{vm} to denote the cost of waste caused by one *VM unit*. The R function is defined as Equation 4.

$$R(v^{(t)}, \hat{v}^{(t)}) = max(0, \hat{v}^{(t)} - v^{(t)})R_{vm}. \tag{4}$$

Combining Equation 3 and Equation 4, the total resource prediction error is quantified as Equation 5.

$$\begin{aligned} C &= f(v^{(t)}, \hat{v}^{(t)}) = \beta P(v^{(t)}, \hat{v}^{(t)}) + (1-\beta)R(v^{(t)}, \hat{v}^{(t)}) \\ &= \begin{cases} \beta v^{(t)} T_{avail} + (1-\beta)(\hat{v}^{(t)} - v^{(t)})R_{vm}, & \text{if } \hat{v}^{(t)} \geq v^{(t)} \\ \beta(\hat{v}^{(t)} T_{avail} + (v^{(t)} - \hat{v}^{(t)})T_{delay}), & \text{if } \hat{v}^{(t)} < v^{(t)} \end{cases} \end{aligned} \tag{5}$$

For different clouds, the trade-off between SLA penalty and idled resources can be different. We use the parameter β to tune the importance between P function and R function. As mentioned in Section II, we aim to minimize the total cost quantified by Equation 1.

B. Ensemble Prediction Model

To incorporate the prediction power of individual prediction algorithm, we utilize ensemble prediction techniques to handle most of the prediction tasks. The ensemble prediction is motivated by its classification counterpart but it has different property. In prediction scenario, temporal correlation is contained between records, while in classification scenario the data is assumed to be i.i.d (independent and identically distributed) [11]. Moreover, the labels of prediction are continuous, so the scale of the ensemble results should be well controlled.

In ensemble prediction, we employ five different time series prediction techniques as shown in Table I. To combine their prediction result, we propose a weighted linear combination strategy. Suppose the predicted value for predictor $p \in \mathcal{P}$ at time t is $\hat{v}_p^{(t)}$ and its corresponding weight at time t is $w_p^{(t)}$, the predicted value for a certain VM type at time t is

$$\hat{v}^{(t)} = \sum_p w_p^{(t)} \hat{v}_p^{(t)}, \text{ subject to } \sum_p w_p^{(t)} = 1. \tag{6}$$

Initially ($t = 0$), all the individual predictors have the same contribution to the prediction result, i.e. $w_p^{(0)} = \frac{1}{|\mathcal{P}|}$. The weight updating strategy for the prediction based ensemble is also different from the traditional classification based strategy. In classification scenario, the results can only be “correct” or “incorrect”, and the ensemble just needs to increase the weights of those correctly classified classifiers for weight updating. In the prediction scenario, the results are continuous values and the weights of the predictors would directly affect the ensemble result. Therefore the updating strategy should be carefully quantified.

We make use of the difference between each predicted value $\hat{v}_p^{(t)}$ and the real value $v^{(t)}$. In order to update the weights, we calculate the *relative error* $e_i^{(t)}$ caused by predictor i at time t according to

$$e_i^{(t)} = \frac{c_i^{(t)}}{\sum_p c_p^{(t)}} w_i^{(t)}, \tag{7}$$

where $c_i^{(t)}$ (or $c_p^{(t)}$) is the prediction cost of predictor i (or p) computed by the cost functions such as *MAE*, *LSE*, *MAPE* and *CPC* (as described in Section IV-A).

Note that the relative errors cannot be used as the new weights of the predictors since they are not normalized. As the final predicted value is the linear combination of all the results of individual predictors, Equation 8 should be used to normalize the weight.

$$w_i^{(t+1)} = \frac{e_i^{(t)}}{\sum_p e_p^{(t)}}. \tag{8}$$

It is easy to prove that the weight of the best predictor at each time is guaranteed to be increased by this weight update strategy.

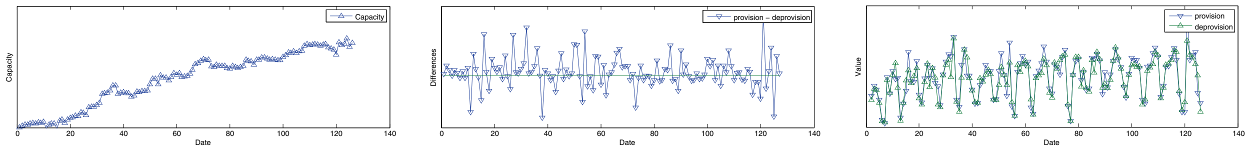
C. Capacity Prediction

1) Time Series Selection: Before adopting ensemble prediction for the problem of capacity prediction, we need to pick a suitable type of time series first. Given the cloud trace data, three types of time series are available for capacity prediction: The original capacity time series (See Figure 5(a)), the capacity change time series (See Figure 5(b)), and the separated provisioning/de-provisioning time series (See Figure 5(c)).

It is easy to observe that the capacity of the cloud gradually increases from the long-term perspective. The non-stationarity property of this time series makes most of the time series prediction models to be invalid. It is true that we can leverage *ARCH* [12] to directly model and predict such non-stationary time series. However, *ARCH* is a parametric model that only performs well under stable conditions. The highly fluctuate request of the cloud service makes the prerequisite of this model unsatisfied. Furthermore, *ARCH* is based on symmetric cost functions. This makes it inappropriate to the cloud scenario.

The capacity change time series is obtained by taking the first derivative of the original capacity time series. This time series shows a stationary trend, and it is proper to be modeled by most of the time series prediction models. However, the irregularity of the trends implies that its temporal pattern is not easy to be discovered.

If we further decompose the capacity change time series into a provisioning component and a de-provisioning component, a weekly periodic pattern appears. This pattern implies that this type of time series is easier than the previous two. In our system, we utilize this kind of time series for prediction. Generally, *the capacity change can be estimated according to the difference between the provisioned and de-provisioned resources*, i.e. $\Delta = prov_t - deprov_t$, where Δ denotes the change of capacity at each time slot, $prov_t$ and $deprov_t$ denote the quantified provisioned and de-provisioned resources.



(a) Original capacity time series

(b) Capacity change time series

(c) Provisioning/de-provisioning time series

Fig. 5. Candidate time series for prediction.

TABLE II
COST MATRIX FOR CAPACITY ESTIMATION

	Provisioning	De-provisioning
Over-estimation	resource waste	SLA penalty
Under-estimation	SLA penalty	resource waste

2) Interpretation of cost functions: Over- and under-estimation have different consequences when estimating the provisioning and de-provisioning of resources. Table II illustrates the cost matrix for these two situations. For provisioning prediction, an over-estimation has no negative effect on the customer and only causes idled resources, but an under-estimation degrades the service quality. For de-provisioning prediction, the costs are reversed.

Cost of SLA penalty (P function): In the scenario of capacity prediction, when there is not enough resource, the requester has to wait until new resources are available. New resources are supplemented by starting new servers at the back-end of the cloud. Therefore, T_{delay} is used to indicate the waiting time plus the physical server cold start time plus the VM fulfillment time, i.e. $T_{delay} = T_{wait} + T_{start} + T_{vm}$. On the other hand, if available resources are enough for current request, the resources associated to the request can be immediately allocated and the provisioning procedure can immediately start. In this situation T_{avail} only include the VM fulfillment time T_{vm}. In our system, we define the P function for provisioning prediction syntactically the same as Equation 3. The P function for de-provisioning prediction can be obtained by switching the variables $v^{(t)}$ and $\hat{v}^{(t)}$.

Cost of resource waste (R function): Syntactically, the R function in the generic cost function is the same as the one for capacity prediction. Semantically, R_{vm} in R function is used to denote the waste of physical server plus the amortized waste of cooling system, electricity supply, and human labor, etc. Also, the R function for de-provisioning prediction can be obtained by exchanging $v^{(t)}$ and $\hat{v}^{(t)}$.

3) A Better Prediction Approach for De-provisioning: The future de-provisioned VM are bounded by the number of total used VMs in the cloud. At any time, any customer cannot de-provision more VMs than they requested. The ensemble technique is workable for de-provisioning prediction, but we have an alternative way to estimate the de-provisioning in a more intuitive way [7]. We will first introduce this method and then show that the new proposed method can achieve better accuracy than the ensemble prediction approach through experiments in Section V.

Since we have all the information of the potential de-provisioned VMs, we can build profiles using the temporal dynamics for the VM image types. Rather than the observed time series, the temporal characteristics provides more interpretable context of the de-provisioning that can facilitate the estimation. Our empirical exploration demonstrate that knowing the current life time of individual VM is helpful for estimating the de-provisioning [8]. That is, we can infer when a certain VM would be de-provisioned through the life time distribution of the images. Since the true distribution of the image life time is not available, we need to estimate it from the historical data first.

To investigate whether the VM life time depends on the time when it is requested, we divide the requests into two groups by putting the requests recorded during Monday and Wednesday to the first group and the remains to the second group. Then we conduct statistical test on these two datasets. The result indicates that we can safely assume that the VM life time distribution does not depend on the time they are requested. Suppose $life(VM)$ is the current life time of a VM, and n_i is the frequency of VMs with life time t_i. We estimate the empirical cumulative distribution function (CDF) of the life time with a step-wise function in Equation 9.

$$\begin{aligned}\hat{F}(x) &= P(life(VM) \le x)\\ &= \begin{cases} n_1/\sum_i n_i, & t_1 \le t < t_2,\\ (n_1+n_2)/\sum_i n_{t_1}, & t_2 \le t < t_3,\\ \dots, & \dots\\ (\sum_{i=1}^{n-1} n_i)/\sum_i n_i, & t_n \le t. \end{cases}\end{aligned} \tag{9}$$

The output of the estimated CDF denotes the probability of a VM that would be de-provisioned when its current life time is t_i. Utilizing $\hat{F}(x)$, the de-provisioning demand can be estimated by

$$\sum_{i \in VM_{active}} \hat{F}(life(i) \le t_{now} - t_{start}(i)), \tag{10}$$

where VM_{active} denotes the set of VMs currently used, t_{now} denotes the current time and $t_{start}(i)$ denotes the provisioned time of VM i.

D. Instant VM Provisioning

In this section, we firstly introduce how we adopt the generic cost function according to the instant VM provisioning scenario. Then we introduce how we reuse the prediction techniques with minor modification to solve the instant VM provisioning problem.

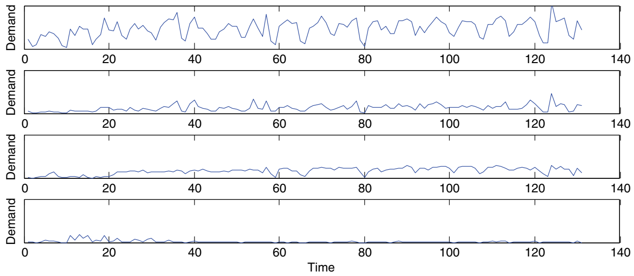


Fig. 6. Total requests time series and individual VM type requests time series.

1) Interpretation of cost function: The cost function for the instant VM provisioning scenario has the same form with the generic cost function, but they have different semantics. The P function and R function represent the *the cost of provisioning delay* and *the cost of idled VM* in the instant provisioning scenario.

When a request for a particular type of VM arrives, if there is no VM with the same type prepared in the cloud pool, the cloud has to provision the VM on-the-fly. We denote the on-the-fly provisioning delay as $T_{delay} = T_{miss}$, which is similar to the "miss" in system cache scenario. On the other hand, if the requested VM has already prepared in the pool, the cloud can simply transfer its ownership to the customer and immediately finish the provisioning procedure. In this case, the request is "hit" and the delay is $T_{avail} = T_{hit}$. Typically, $T_{hit} \ll T_{miss}$. A "hit" request can be handled in seconds, but the on-the-fly provisioning time for a "miss" request could be up to minutes for the state-of-art cloud systems. If the requested VM is configured with complex service components, a missed request would take even longer due to the software integrity, security checking, and sometimes other manual processes.

When the provisioning of a certain VM type is over-estimated, part of the prepared VMs would be wasted. The service vendor would be responsible for the cost of this part of VMs, since they cannot be charged to the customers. Same as in capacity prediction scenario, the cost of an idled VM can be quantified by the number of *VM units* and we can use R_{vm} to denote its cost. Therefore, the over-predicted cost can be quantified by the same equation (Equation 4) used in capacity prediction scenario.

2) Prediction of individual VM provisioning: Different from the provisioning and de-provisioning prediction for capacity prediction, to enable instant VM provisioning, we need to separately handle the VM provisioning time series of each VM type. As shown in Figure 6, the time series of individual type requests demonstrates high irregularity comparing with the total requests time series. This phenomenon implies that the auto-correlation of these time series is more difficult to discover. Therefore, besides the auto-correlation, we also exploit the correlation between different VM types to mitigate the risk of serious prediction deviation. For instance, the requests time series of the same series of VM type, i.e. *Windows Server 2003* and *Windows Server 2005*, are highly correlated. We called this procedure *Correlation Ensemble*.

In our system, we utilize the correlation matrix of time series to help post-process the prediction results. Suppose $\sum_{(t)}$ is the covariance matrix of VM types at time t, $cov_{ij}^{(t)}$ denotes the covariance between resource types i and its jth correlated resource. By considering the positive influence of the strongly correlated time series (in our system, we set correlation as 0.8, based on the empirical study), the prediction value $\hat{u}_i^{(t)}$ of time series i at time t now becomes

$$\hat{u}_i^{(t)} = \frac{\sum_{j=1}^{k} cov_{ij}^{(t-1)} s_{ij} \hat{v}_k^{(t)}}{\sum_{j=1}^{k} s_{ij} cov_{ij}}, \tag{11}$$

where $s_{ij} = \bar{t_i}/\bar{t_j}$ denotes the scale difference between two time series and k is the number of strongly correlated time series. In our current design, we only considered the positively correlated time series, the negative influence consideration is one of our future works.

To further mitigate the cost of mis-prediction caused by the inherent prediction difficulty, we introduce a module called *Reservation Controller*. Its function is to reserve the unused VMs and only notifies the cloud to prepare new VMs when all the reserves of the VM type are used up. Reservation Controller provides a good buffer mechanism that effectively reduces the waste of VMs. It can be integrated into the cloud configuration module.

V. Experimental Evaluation

To evaluate the effectiveness of our system, we use the real VM trace log of IBM Smart Cloud Enterprise (SCE) to conduct the experiments. The trace data we obtained records the VM requests for more than 4 months (from late March 2011 to July 2011), and it contains tens of thousands of request records with more than 100 different VM types. In this trace data, each request record contains 21 features such as *Customer ID*, *VM Type*, *Request Start Time*, and *Request End Time*, etc. The goal of the experimental evaluation is to answer the following questions:

- Whether our prediction mechanism for capacity planning and instant VM provisioning reliable?
- Whether the measure *CPC* is practical and flexible?
- To what extent can our system decrease the average request fulfillment time for both problems?

A. Data Filtering and Aggregation

Data preprocessing is the step before data modeling and prediction. There are two reasons for data preprocessing of the raw trace data recorded by SCE.

1) The raw data contains useless request fields that would not be used during prediction. Also, it contains the internal test requests that brings noise during temporal pattern mining.
2) The records of the requests are stored in a low-level representations. The requests need to be aggregated into proper granularity first, and then feed to the algorithm for prediction.

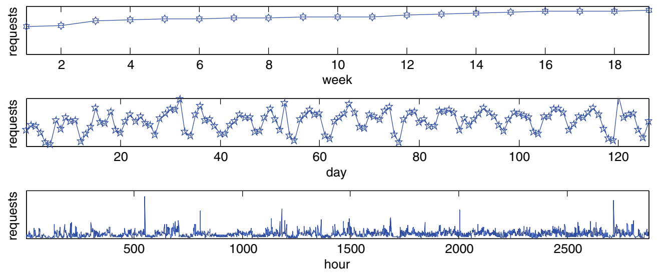


Fig. 7. Time series with different granularities. From top to bottom, the time series are aggregated by week, day, and hour, respectively.

TABLE III
STATISTICS FOR TIME SERIES IRREGULARITY

Measure Granularity	Week	Day	Hour
Coefficient of Variance	0.1241	0.4048	0.7182
Skewness	-0.5602	-0.2536	2.6765
Kurtosis	2.7595	2.5620	20.5293

1) Feature Selection: Not all the 21 features features of a request record are useful. In our current implementation, we only consider the *VM Type*, which illustrates the type of VM the customer requests; *Request Start Time*, which indicates the time that the customer sends the request; *Request End Time*, which indicates when the VM is released. Other features like *Customer ID*, *Customer Priority* and *Data Center* will be considered in our future work for personalized service quality improvement.

2) Time Series Aggregation Granularity Selection: All the time series we present in this paper are obtained through aggregating the raw trace records with a certain granularity. The time series aggregated by different granularities would have different levels of difficulty for prediction. For example, Figure 7 shows the capacity provisioning time series aggregated by week, day, and hour, respectively.

This figure shows that the coarser the granularity, the larger the provisioning amount in each time slot. Therefore, the weekly aggregated time series requires the cloud to prepare the most VMs for each time slot. Compared with a finer granularity, a smaller portion of prediction deviation for weekly aggregated time series would result in a larger waste. Moreover, through exploration, we found the life time of most of the VMs is shorter than one week. Therefore, the weekly aggregated time series cannot reflect the real situation.

On the contrary, it is also not suitable to aggregate the records by hour, since the lifetime of the VMs is not so short. A too fine granularity would make the value on each timestamp lacks statistical significance. Table III list the results by measuring the irregularity of these time series in different perspectives with *Coefficient of Variance (CV)*, *Skewness*, and *Kurtosis* [13]. Higher *CV* indicates larger volatility, higher *Skewness* indicates stronger asymmetry, and higher *Kurtosis* means more of the variance is the result of infrequent extreme deviations. In our comparison, the time series aggregated by hour has the largest values in all the three measures, indicating the hour granularity is not suitable to aggregate the time series. Therefore, based on above investigation, we aggregate the daily time series in our implementation.

3) VM Type Selection: VM type selection is only relevant to instant VM provisioning problem. As mentioned in Section IV-D, we need to predict the future demand for each VM type separately. Figure 8(a) plots the distribution of requests in a time-type-request view. This figure clearly indicates one obvious characteristics: The distribution of VM request is highly uneven, and a small number of the VM types dominate the distribution. We also plot the corresponding cumulative distribution function (CDF) and rank the VM based on their requests frequency in Figure 8(b) and Figure 8(c), respectively. The CDF shows that the VM requests obey the 80/20 rules — *more than 80% of the requests concentrates on less than 20% of the VM types*. In the frequency ranking plot, we observe that there is an inflection point between the 12th and 13th types, which explicitly divides the types into frequent and infrequent groups. The measures like *CV*, *Skewness* and *Kurtosis* on time series of these infrequent types also show higher values than those of frequent types, which demonstrate that the time series of these infrequent types are not regular enough to be modeled and predicted.

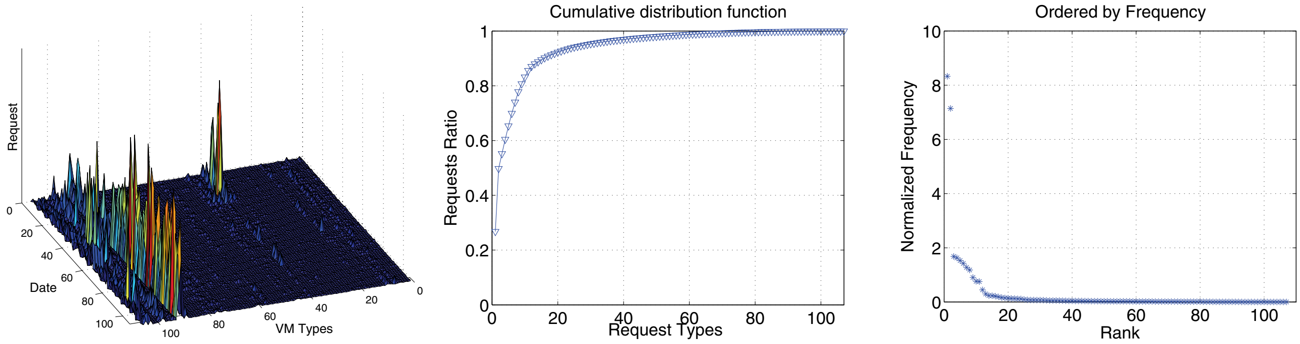
Notwithstanding the inherent prediction difficulty caused by the irregularity, the future demand of the infrequent VM types can be handled in a more empirical way. We design the data preprocessing module to periodically checks the change of the rank, and build prediction modules for the frequent VM types only. For the infrequent types, the cloud prepare a fixed number of VMs for each type. The fixed number is set as the moving average of the recent requests of the type. Once the prepared VMs are used up, the cloud just needs to prepare another batch of VMs.

B. Provisioning Prediction Evaluation

To evaluate the performance of provisioning prediction, we compare the accuracy of our ensemble predictor with the individual predictors mentioned in Table I. In this experiment, we use *CPC* as the measure. For parameter setting, we set $\beta = 0.5$, $T_{delay} = 1200$ (the time for on-demand preparation, including server boot up, VM creation, security checking, patching, and configuration etc.), $T_{avail} = 10$ (resource is available almost instantly), and $R_{vm} = 500$ (the cost of wasted resources for one VM unit).

We partition the time series horizontally into two parts: the records before May 2011, and the records after May 2011. Then we use the data before May 2011 for training. The precision of these predictors are evaluated on the month of May, June, and July, respectively.

For each individual predictor, the parameter setting is as follows. *Random Guess* simply guesses the future demand by randomly picking a number between 0 and the maximum demand known so far. We set the sliding-window of *MA* and the number of variables in *AR* as 7, since we have observed a suspected weekly periodical pattern. For *Neural Network*, we leverage a 3-layer topology with 1 hidden layer. The neurons in the input layer take the information of the latest 7 days as input variables, and the number of hidden layer and output layer is set as 7 and 1, respectively. This topology enables the neural network to capture any combinations of the input variables. As for *SVM*, we first leverage the grid search to



(a) Request distribution in time-type-request view

(b) CDF of VM Types

(c) CDF of VM Types

Fig. 8. Distribution about requests.

TABLE IV
THE COST OF DIFFERENT PREDICTORS MEASURED BY *CPC*

Predictor	May	June	July	Average
Random Guess	2281555	3507600	3080320	2956491
Moving Average	1295550	1293620	1293620	1294263
Auto Regression	504912	760110	1047275	770765
Artificial Neural Network	980780	1095102	1577127	1217669
Gene Expression Programming	866117	640405	1037705	848075
Support Vector Machine	3746005	2199010	1147240	2364085
Ensemble	538302	626585	1072840	**745909**

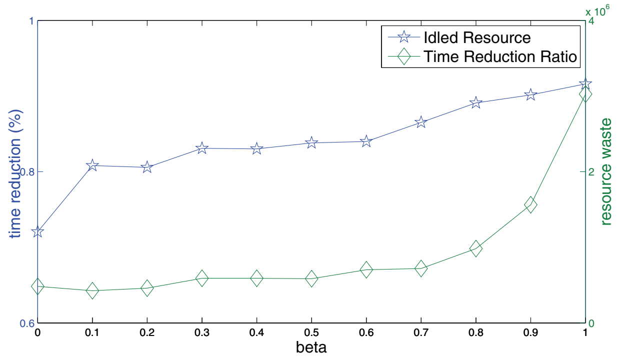


Fig. 9. The effect of tuning β for capacity prediction.

identify the best parameter combinations with training dataset, and then we use regression *SVM* for prediction. For *GEP*, we set the population as 40, the number of evolution generation as 1000, the operator set as $\{+, -, \times, /, \sqrt{}, \sin\}$. To eliminate the randomness of some predictors (*Random Guess*, *Artificial Neural Network* and *Gene Expression Programming*), the results are computed by averaging 10 runs of each predictor. Table IV lists the evaluation results of all the predictors. The results clearly show that the best predictor is different for different test datasets. And on average, the ensemble method achieves the best performance.

1) Effectiveness of Parameter Tuning: As mentioned before, the parameter β can be used to tune the preference of the predictor (optimistic vs. pessimistic). A higher β results in less SLA penalty risk (more time reduction) but increases the chance of resources waste. A lower β can reduce the idled resources risk but increases the chance of SLA violation.

Figure 9 illustrates how β affects the prediction results. In this figure, the x-axis denotes the value of β while y-axis denotes the average ratio of VM fulfillment time reduction (left) and the average cost of idled resource (right) each day quantified by R function.

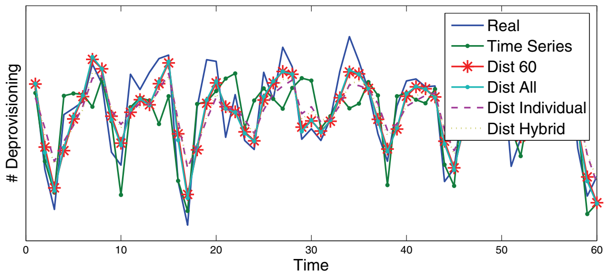


Fig. 10. Prediction result of de-provisioning time series.

As is shown, alone with the increasing of β, both the time reduction ratio and the cost of idled resource cost increase. This is because when β is small (close to 0), the R function would have higher impact to the cost function than P function. In this case, the cloud would prepare relatively less resources to avoid the waste and does not take too much action to reduce the waiting time. On the contrary, when β is big (close to 1), the cost function will pay more attention to reduce the VM fulfillment time. In this case, the cloud would prepare relatively more resources to ensure the resource supply on the cost of higher chance of resource wasting.

C. De-provisioning Prediction Evaluation

We try four different methods to predict the de-provisioning demands based on the information of VM life time. For all these methods, the evaluation is conducted on the last 60 days of de-provisioning records, and all the other data are used as the training data. The details of these methods are listed as follows:

1) *Dist All*: This method leverages all the training data to estimate the global VM life time distribution by ignoring the VM type. For each day, it estimates the expected number of VMs that would be de-provisioned based on the probability of de-provisioning.
2) *Dist 60*: This method leverages the latest 60 days of training data to estimate the global VM life time

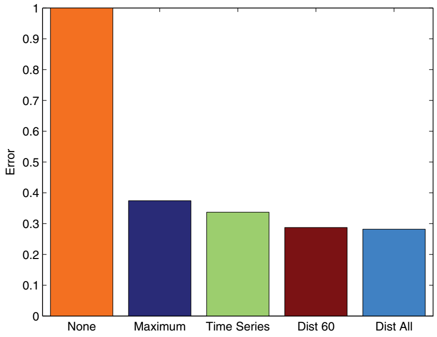


Fig. 11. Errors of different methods (normalized).

distribution by ignoring the VM type. For prediction, it is the same as the first method.

3) *Dist Individual*: This method first estimates the life time distributions of each VM type, and then it predicts the expected number of de-provisioning VMs based on the life time distributions of corresponding types. This method is a finer granularity version of the first method.
4) *Dist Hybrid*: This method leverages all the training data to estimate both the global life time distribution and individual life time distribution. The expected number of de-provisioned VMs is calculated as $\alpha \hat{F}_i + (1-\alpha)\hat{F}_g$, where α equals to the fraction between the frequency of specified image type and the most popular image type.

Besides these four methods, we also predict the future de-provisioning with the ensemble method and their prediction results (6 methods in total) are shown in Figure 10, where the x-axis denotes the time and y-axis denotes the number of de-provisioned VMs. As is illustrated, all of these methods successfully discover the periodic patterns of the de-provisioning trend. However, the ensemble method has limited ability to fit the scale of the peaks. Among all these methods, the time series predicted by *Dist All* fits the real time series the best. Moreover, all these four life time distribution based methods have approximately the same accuracy and they all outperform the ensemble prediction method. The reason is that the methods *Dist All*, *Dist 60*, *Dist Individual*, and *Dist Hybrid* all estimate the de-provisioning from the empirical life time distributions, rather than simple inference from the observed time series.

We also use *CPC* to quantify the errors of the ensemble time series prediction and two best distribution based prediction method – *Dist 60* and *Dist All*. To better demonstrate the effectiveness of the prediction methods, we also calculate the cost of two naive method: *None* and *Maximum*. *None* takes no action for capacity planning. All the requests are responded by preparing the resource on-the-fly. *Maximum* always prepares the maximum capacity for the cloud. Figure 11 shows the evaluation result after normalization. It clearly shows that any method that takes the pre-action is significantly better than *None*. Moreover, all three sophisticated methods are better than *Maximum*. Among all these methods, *Dist All* makes least error among all the methods.

D. Instant VM Provisioning Evaluation

To evaluate the effectiveness of request prediction for individual VM type, we compare the prediction performance of individual predictors with our ensemble predictor. Besides *CPC*, we also use *MAP*, *MSE* and *MAPE* to measure the precision of these predictors.

For VM type, we pick the top 12 (before the inflection point in Figure 8(c)) most frequent VM types for experiments. All the time series are partitioned into two sets, the records for the last 30 days are used as the test data, while the remaining are used as the training data. Similar to the experiment in provisioning evaluation, grid search is utilized to seek the best parameter combinations for individual predictors.

Table V shows the precision of all the predictors on all the time series. Due to the space limit, we only list the details of the top 3 time series (*Red Hat Enterprise Linux 5.5 32-bit*, *Red Hat Enterprise Linux 5.5 64-bit*, and *SUSE Linux Enterprise Server*) and the average *CPC* of all the time series. It can be easily observed that the best predictor is different for different VM types. For example, GEP performs the best on the 1st VM type; ANN achieves good results on the 2nd VM type. Moreover, the winner predictor of one VM type can also perform badly for other VM types. For example, ANN obtains a poor precision on the 1st VM type.

For our ensemble predictor, although it does not perform the best on any single VM type, it is very robust as its performance is always close to the winner predictor on all the types. The average *CPC* shows that our ensemble predictor has the best average performance, indicating its self-adaptation capability.

Figure 12 displays the real time series of the three most frequent VM types and their corresponding prediction results of all the predictors. In this figure, the ensemble predictor can always identify the best predictor for the time series and quickly converge to it. Since under-prediction is worse than over-prediction in cloud provision scenario, the predictor that rarely under-predicts the demands is considered better than the one whose outputs are always close to but less than the real demands. It can also be noted that although MA and SVM do not have the best performance in either of the three VM types, they can also make contributions to the ensemble predictor according to their weights.

1) Detailed Cost: To better investigate the composition of cost, we also calculate the prediction cost for each component.

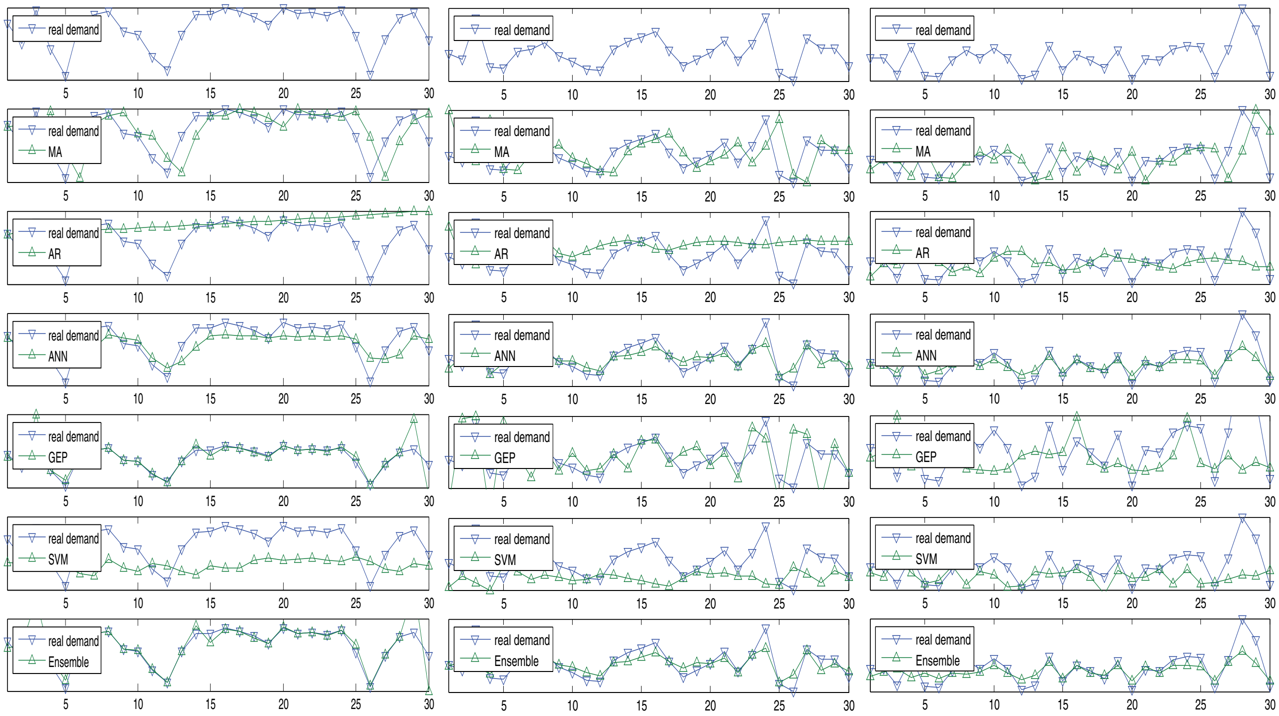
Provisioning time reduction: The most important criterion to evaluate the performance is how much provisioning time can be saved. We calculate the proportion of time reduction obtained by predictors based on (12), where the save portion (p_{save}) is calculated according to Equation 12:

$$p_{save} = \frac{\sum_t (max(v^{(t)} - \hat{v}^{(t)}, 0)T_{miss} + \hat{v}^{(t)}T_{hit})}{\sum_t v^{(t)}T_{miss}}. \quad (12)$$

Figure 13(a) shows the proportion of time reduction of each predictor. It is good to observe that most of the predictors can significantly decrease more than 60% of the provisioning time. However, in the presence of large variations across time series, the saved time achieved by the predictor is not stable for different VM types. On average, our ensemble predictor performs the best due to its strong self-adaptation ability.

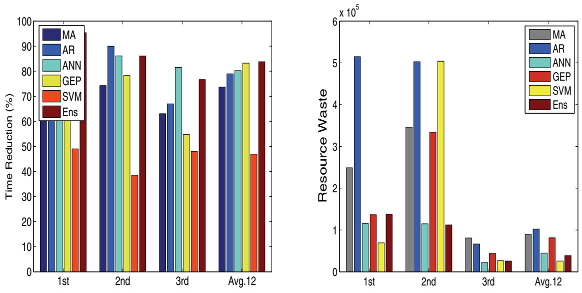
TABLE V
THE COST OF PREDICTION ALGORITHMS UNDER DIFFERENT MEASUREMENTS. FOR THE ACRONYM, *MA* DENOTES *Moving Average*, *AR* DENOTES *Autoregression*, *ANN* DENOTES *Artificial Neural Network*, *GEP* DENOTES *Gene Expression Programming*, AND *SVM* DENOTES *Support Vector Machine*

Predictor	1st VM Type				2nd VM Type				3rd VM Type				Avg. Top 12
	MAE	MSE	MAPE	CPC	MAE	MSE	MAPE	CPC	MAE	MSE	MAPE	CPC	CPC
MA	40.3	2740.23	0.87	437305	51.53	4904.4	1.63	532132.5	7.23	88.5	1.67	76370	152402.29
AR	45.1	4034.57	1.52	300687.5	50.5	3893.63	5.32	391630	5.9	67.5	1.05	83292.5	137046.67
ANN	30.27	1203.07	0.67	388535	20.8	706.4	2.04	247085	4.97	29.57	2.14	37052.5	106113.96
GEP	17.37	1757.5	0.21	154440	54.77	5621.97	6.59	473347.5	4.1	27.63	1.59	33745	112149.79
SVM	68.03	5604.77	0.9	1010422.5	59.5	5578.37	2.89	890447.5	6.2	85.4	0.9	88737.5	234225.83
Ensemble	16.7	1212.1	0.21	158862.5	21.67	1057	2.04	254190	5.03	46.97	1.56	53327.5	88679.79



(a) Red Hat Enterprise Linux 5.5 (32-bit) (b) Red Hat Enterprise Linux 5.5 (64-bit) (c) SUSE Linux Enterprise Server

Fig. 12. Prediction results on testing data of three most frequent time series. ▽ denotes the original time series while △ denotes the predicted time series. From top to bottom, the time series are: (1) The real time series; (2) Time series predicted by MA; (3) Time series predicted by AR; (4) Time series predicted by ANN; (5) Time series predicted by GEP; (6) Time series predicted by SVM; (7) Time series predicted by Ensemble.



(a) Average Provisioning Time Reduction (b) Average Resource Waste

Fig. 13. Average provisioning time reduction and average resource waste.

2) Idled VMs: The cost of idled resources is another evaluation criterion of the quality of prediction. It is true that an always over-predicted predictor can save a lot of provisioning time, but such a predictor would also waste a lot of resources. Figure 13(b) shows the amount of idled resource caused by each predictor. On average, the best resource saver is SVM, but its performance in time reduction is the worst. Also note that GEP achieves a good performance on time reduction, but it wastes the resources twice as much as our method.

3) Effectiveness of Reservation Controller: Table VI shows the ratio of provisioning time reduction that can be achieved by incorporating the *Reservation Controller*. For all the time series, *Reservation Controller* further improves the reduction of the average provisioning time from 83.79% to 94.22%. Moreover, with the assistance of reservation controller, the over-prediction portion of the VMs prepared before can be used for following days without going through the provisioning process again.

4) Effect of Sophisticated Cost Functions: As mentioned before, our proposed framework is flexible and can use different cost functions to guide the prediction process. In this section, we explore how the prediction is influenced by the use of complex cost functions.

For instance, in practice, a fixed amount of resource R_{fix}, i.e. the standing resources, is always provided for VMs prepa-

TABLE VI
PROVISIONING TIME REDUCTION BY INCORPORATING RESERVATION CONTROLLER

VM Type	1	2	3	4	5	6
Reduction	97.77%	91.58%	96.81%	96.26%	96.76%	91.67%
VM Type	7	8	9	10	11	12
Reduction	98.17%	94.37%	99.56%	85.35%	84.98%	97.45%

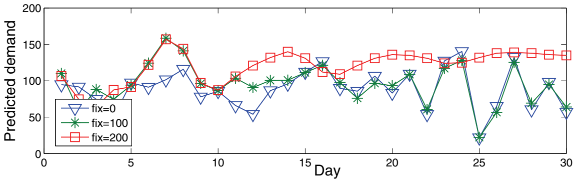


Fig. 14. The influence of R_{fix} on prediction (normalized data).

ration. If the over-predicted value lies below such a threshold, the cost of resource is still 0. Equation 13 shows the form of this cost function:

$$R(v^{(t)}, \hat{v}^{(t)}) = \begin{cases} 0 & \text{if } \hat{v}^{(t)} < min(R_{fix}, v^{(t)}), \\ (\hat{v}^{(t)} - R_{fix})R_{vm} & \text{if } v^{(t)} < R_{fix} < \hat{v}^{(t)}, \\ (\hat{v}^{(t)} - v^{(t)})R_{vm} & \text{if } R_{fix} \leq v^{(t)} < \hat{v}^{(t)}. \end{cases} \tag{13}$$

Figure 14 shows three predicted demands time series time generated by different R_{fix} values. As R_{fix} increases, the predicted demand tends to be more optimistic than pessimistic. Experimental results show that the average waiting time can be reduced by 93.94% when R_{fix} increases to 200, but the amount of idled resources becomes huge.

E. Comparison of Different Measures

Table VII shows the time reduction and the idled resources of the ensemble predictor guided by various cost measures. *CPC* clearly outperforms the other three cost measures in provisioning time reduction. We also find that *CPC* has the largest idled resources. Such a phenomenon can be well interpreted by the basic idea of these cost measures. *MAE*, *MSE* and *MAPE* are all symmetric cost measurements and they guide the ensemble predictor to equally weight the under-predictors and over-predictors. While *CPC* gives more penalty to under-predictors than over-predictors, the ensemble predictor always prefers to giving larger weights to the over-predictor, which results in more time reduction and also more likely to waste resources. As in cloud service scenario, customer service quality is much more important, it is worth reducing the provisioning time on the costs of a reasonable amount of idled resources.

F. Model Computational Cost

Our proposed method is neither computationally intensive nor storage intensive. In terms of CPU cost, given the historical requests (about tens of thousands per month), the training time costs less than 1 minute. Once the training is finished, the prediction can be conducted constantly (within 1 second).

The consumption of memory cost can also be ignored. This is because the only thing need to be put in memory is the parameters of these algorithms. For example, the number of parameters in *Neural Network* is $\sum_{i=1}^{L} s_{i-1} * s_i$, where s_i denotes the number of neurons in level i. Therefore, the total storage cost of the proposed method is no more than 1000.

TABLE VII
WAITING TIME REDUCTION AND RESOURCE WASTE OF PREDICTOR GUIDED BY DIFFERENT COST MEASUREMENTS

Time Reduction	Avg. Time Reduction	Avg. Resource Waste
MAE	81.77%	28300
MSE	80.13%	37100
MAPE	79.01%	33133
CPC	83.79%	38533

In terms of scalability, the growth of cloud would only increase the value of requests at each timestamp, it would not increase the scale of the input data of the proposed algorithm.

VI. RELATED WORKS

A. System Oriented Data Mining

The increasing complexity and scale of modern computing systems make the analytic difficulty far beyond the capability of human being. The emergence of system auxiliary technologies liberates the heavy burden of system administrators by leveraging the state-of-art data mining technologies. There are mainly two types of system auxiliary technologies: the system analytical techniques and the system autonomous techniques. For the first type, [14][15][16] utilized text mining to find out the category of events and then exploit visualization techniques to show the results. [17][18][19][20][21] utilized temporal mining and encoding theory to discover the event interaction behaviors from systems logs and then summarizes them in a brief way. Xu et al [22][23][24] focus on proposing anomaly detection algorithms to detect the system faults by utilizing the system source code and logs. These above methods are all off-line algorithms and are unable to tell the system to take reactions on-the-fly. For the second type, [25] used motif mining to model and optimize the performance of data center chillers. [26] proposed a signature-driven approach for load balance in the cloud environment with the help of utilization data. Our proposed solution can be categorized as an system autonomous technique. Different from existing works, the capacity planning and instant VM provisioning problems we face are related to the whole cloud environment scale.

B. Behavior Modeling and Prediction for System

In operating system, caching is one of the common techniques used to improve the system performance through forecasting. *Partitioned Context Modeling* (PCM) [27] and *Extended Partitioned Context Modeling* (EPCM) [28] are two statistical based caching algorithms, which model the file accesses patterns to reliably predict upcoming requests. These two methods have shown much better performance over the traditional caching algorithms like *Least Recent Used* (LRU), *Least Frequent Used* (LFU) and their extensions. While in the area of modern large-scale system behavior prediction, there is only little related works. Different from the traditional shared memory scenario, the virtual machines in the cloud environment can not be shared and reused due to the security issue. Therefore, there should be multiple copies of virtual machines prepared for multiple requests. The work of [1] also studied the cloud resource prediction problem. But it only focuses on predicting the VM resource (CPU, memory, etc) for individual VMs. For our work, rather than predicting the resource usage within VMs, we aim to predict the capacity and VMs demands for the whole cloud. AutoScale [29] proposes the control policy on data center capacity management. Their method mainly focus on reducing the SLA violation when the bursts come, instead of improving the expected provisioning accuracy. The work of CloudScale [30] is a good compliment of our work. While our work focusing on resource prediction from the macro perspective (at the day level), they focus on prediction adjustment on the micro perspective (at the minute and second level).

C. Virtual Environment Management

Virtual environment management technologies are mainly based on control theory, meaning that the actions are taken after the event happens. Auto-scaling proposed by Amazon [31] is the customer side auto-scaling management component, which allow customers to set up their own rules to manage the capacity of their virtual resources. This method focuses on the customer side post-processing of capacity tuning rather than the provider side. For the provider side, [6] proposed a self-adaptive system based on temporal data mining to reduce the VM provisioning time. In [32], the resources allocation problem is modeled as a stochastic optimization problem with the objective of minimizing the number of servers as well as the SLA penalty. [33] proposed a technique to enable the auto-scaling of visualized data center management servers. All these works are focusing on the resource allocation and scheduling with a fixed amount of resources, while our work aims at estimating the total amount of resources for the whole cloud environment.

VII. Conclusion and Future Work

In this paper, we discuss two data mining based analytical techniques to improve the cloud service quality. Specifically, we believe that both the capacity planning and the instant VM provisioning problems can be handled by prediction, where the first problem can be solved by preparing the available resources beforehand and the second problem can be solved by pre-provisioning the needed VM instances. According to the unique characteristic of cloud service, we propose a novel cost-sensitive measure called *CPC* to guide the prediction procedure. To demonstrate the effectiveness of our approach, we implement a prototype and conduct a series of simulation experiments based on the trace data of IBM Smart Cloud Enterprise. The experimental evaluation results demonstrate that our approach is able to effectively improve the service quality while retaining a low resource cost.

There are several promising directions for our future works. First, only a subset of the features of the trace records is used to build the prediction model in our current work. We believe that we can further improve the service quality by incorporating more features into the prediction model. Second, we can further improve the prediction accuracy by providing customer-oriented personalized prediction. Such a target can be fulfilled by leveraging the customer profile information. Third, our current solution for capacity planning is a kind of mid-term prediction. The irregularity of small granularity time series makes the prediction inherently difficult. To conquer this difficulty, we consider to leverage the techniques in control theory to conduct the short-term capacity planning. Finally, we can reasonably assume that some VM requests are the scale-out attempts to reduce workload from the overloaded VMs. So by monitoring resource utilization of individual VMs, we can predict the imminent provisioning requests and prepare the right types of VMs in advance.

Acknowledgement

The work is partially supported by National Science Foundation (NSF) under grant IIS-0546280, HRD-0833093 and CNS-1126619 and by Army Research Office (ARO) under grant W911NF-12-1-0431.

References

[1] Z. Gong, X. Gu, and J. Wilks, “Press: predictive elastic resource scaling for cloud systems,” in *2009 CNSM*.

[2] J. Hamilton, “Cooperative expendable micro-slice servers (CEMS): low cost, low power servers for internet-scale services,” in *2009 CIDR*.

[3] A. Greenberg, J. Hamilton, D. Maltz, and P. Patel, “The cost of a cloud: research problems in data center networks,” *Computer Commun. Review*, 2009.

[4] F. Labonte, P. Mattson, I. Buck, C. Kozyrakis, and M. Horowitz, “The stream virtual machine,” in *2004 ICPACT*.

[5] H. A. Lagar-Cavilla, J. A. Whitney, A. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan, “Snowflock: rapid virtual machine cloning for cloud computing,” in *2009 EuroSys*.

[6] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, “ASAP: a self-adaptive prediction system for instant cloud resource demand provisioning,” in *2011 ICDM*.

[7] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, “Intelligent cloud capacity management,” in *2012 NOMS*.

[8] Y. Jiang, C.-S. Perng, T. Li, and R. Chang, “Self-adaptive cloud capacity planning,” in *2012 SCC*.

[9] I. S. C. Enterprise. Available: http://www-935.ibm.com/services/us/igs/cloud-development/.

[10] C. Chatfield, *The Analysis of Time Series: An Introduction*, 6th ed. Chapman and Hall, 2003.

[11] T. Hastie, R. Tibshirani, and J. I. Friedman, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd ed. Springer, 2009.

[12] R. Engle, “Autoregressive conditional heteroscedasticity with estimates of variance of united kingdom inflation,” *Econometrica*, 1982.

[13] G. McCabe and D. Moore, *Introduction to the Practice of Statistics*. W.H Freeman and Company, 2005.

[14] W. Peng, T. Li, and S. Ma, “Mining log files for data-driven system management,” *SIGKDD Explorations, Volume 7*, vol. 1, pp. 44–51, 2005.

[15] T. Li, W. Peng, C.-S. Perng, S. Ma, and H. Wang, “An integrated data-driven framework for computing system management,” *IEEE Trans. Systems, Man, and Cybernetics, Part A*, vol. 40, no. 1, pp. 90–99, 2010.

[16] T. Li, F. Liang, S. Ma, and W. Peng, “An integrated framework on mining logs files for computing system management,” in *Proc. 2005 KDD*, pp. 776–781.

[17] J. Kiernan and E. Terzi, “Constructing comprehensive summaries of large event sequences,” in *Proc. 2008 KDD*.

[18] J. Kiernan and E. Terzi, “Constructing comprehensive summaries for large event sequences,” *ACM Trans. Knowledge Discovery from Data*, 2009.

[19] W. Peng, H. Wang, M. Liu, and W. Wang, “An algorithmic approach to event summarization,” in *Proc. 2010 SIGMOD*.

[20] W. Peng, C.-S. Perng, T. Li, and H. Wang, “Event summarization for system management,” in *Proc. 2008 KDD*.

[21] Y. Jiang, C.-S. Perng, and T. Li, “Natural event summarization,” in *2011 CIKM*.

[22] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *2009 SOSP*.

[23] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Online system problem detection by mining patterns for console logs," in *2009 ICDM*.

[24] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Mining console logs for large system problem detection," in *2008 SysML*.

[25] D. Patnaik, M. Marwah, R. Sharma, and N. Ramakrishnan, "Sustainable operation and management of data center chillers using temporal data mining," in *2009 KDD*.

[26] Z. Gong, P. Ramaswamy, X. Gu, and X. Ma, "SIGLM: signature-driven load management for cloud computing infrastructures," in *2009 IWQoS*.

[27] T. M. Kroeger and D. D. Long, "The case for efficient file access pattern modeling," in *Proc. 1999 HotOS*.

[28] T. M. Kroeger and D. D. Long, "Design and implementation of a predictive file prefetching algorithm," in *2002 USENIX*.

[29] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch, "Autoscale: dynamic, robust capacity management for multi-tier data center," *ACM Trans. Computer Systems*, 2011.

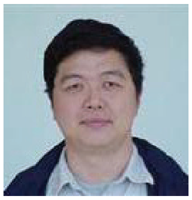
[30] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *2011 SoCC*.

[31] A. AutoScaling. Available: http://aws.amazon.com/autoscaling/.

[32] S. Mylavarapu, V. Sukthankar, and P. Banerjee, "An optimized capacity planning approach for virtual infrastructure exhibiting stochastic workload," in *2010 SAC*.

[33] S. Meng, L. Liu, and V. Soundararajan, "Tide: achieving self-scaling in virtualized datacenter managment middleware," in *2010 MiddleWare*.

Yexi Jiang is currently a Ph.D. student in the School of Computer Science at Florida International University. He received B.S. and M.S. degree in Computer Science from Sichuan University in 2007 and 2010, respectively. His research interest includes system oriented data mining, intelligent cloud, large scale data mining, and semantic web.



Chang-Shing Perng is a research staff member in IBM T. J. Watson Research Center. He received his Ph.D. degree in computer science in 2000 from the University of California, Los Angeles, and has been at IBM since then. His current research interests include temporal data mining, autonomic computing and intelligent system management design.

Tao Li is currently an associate professor in the School of Computer Science at Florida International University. He received his Ph.D. degree in Computer Science in 2004 from the University of Rochester. His research interests are in data mining, computing system management, and information retrieval. He is a recipient of USA NSF CAREER Award and multiple IBM Faculty Research Awards.

Rong N. Chang is Manager and Research Staff Member at the IBM T.J. Watson Research Center. He received his Ph.D. degree in computer science and engineering from the University of Michigan at Ann Arbor in 1990 and his B.S. degree in computer engineering with honors from the National Chiao Tung University in Taiwan in 1982. Before joining IBM in 1993, he was with Bell Communications Research. He is a holder of the ITIL Foundation Certificate in IT Services Management. His accomplishments at IBM include the completion of a nomination-based Micro MBA Program, one IEEE Best Paper Award, and many IBM awards, including four corporate-level Outstanding Technical Achievement Awards and six division-level accomplishments in the areas of cloud computing, IT infrastructure Healthcheck, SLA and business service management, monitoring, and event management, and e-commerce. He is an Associate Editor of the IEEE TRANSACTIONS ON SERVICES COMPUTING. He has chaired several conferences and workshops in cloud computing and Internet-enabled distributed services and applications. He holds 20+ patents, and has published 40+ refereed technical papers at reputable international conferences and journals. He is a Senior Member of ACM and IEEE, and a member of Eta Kappa Nu and Tau Beta Pi honor societies.