Optimal Cloud Resource Auto-Scaling for Web Applications

Conference Paper · May 2013

DOI: 10.1109/CCGrid.2013.73

Optimal Cloud Resource Auto-Scaling for Web Application

Jing Jiang, Jie Lu, Guangquan Zhang, Guodong Long
DeSI Lab, Centre for Quantum Computation & Intelligent Systems
School of Software, Faculty of Engineering and Information Technology
University of Technology Sydney, Australia
Email: {jing.jiang-1, guodong.long}@student.uts.edu.au,
{jie.lu, guangquan.zhang}@uts.edu.au

***Abstract*—In the on-demand cloud environment, web application providers have the potential to scale virtualized resources up or down to achieve cost-effective outcomes. True elasticity and cost-effectiveness in the pay-per-use cloud business model, however, have not yet been achieved. To address this challenge, we propose a novel cloud resource auto-scaling scheme at the virtual machine (VM) level for the web application providers. The scheme automatically predicts the number of web requests and discovers an optimal cloud resource demand with cost-latency trade-off. Based on this demand, the scheme makes a resource scaling decision that is up or down or NOP (no operation) in each time-unit re-allocation. We have implemented the scheme on Amazon cloud platform and evaluated it using three real-world web log datasets. Our experiment results demonstrate that the proposed scheme achieves resource auto-scaling with an optimal cost-latency trade-off, as well as low SLA violations.**

***Keywords*-Cloud computing; Elastic Computing; Resource prediction; Resource scaling; Web services;**

I. Introduction

The popularity of on-demand cloud service spurs the migration of increasing numbers of web applications to the cloud. One of the most attractive aspects for cloud web application providers is the ability to access computing resource elastically (by scaling up or down) according to dynamic resource demands. In this scenario, providers only pay for resources that are consumed at a specific point in time which if operated correctly, will result in less cost and higher quality service (e.g. latency/response time, frames per second during a video) than is achievable by hosting on standard hardware [1]. In a classical case in April 2008, Animoto, an image-processing web applicaiton, experienced a demand jump from 50 instances to 4000 instances (Amazon EC2 instances) in just three days; following the peak, traffic fell sharply to a normal level that was well below the peak [2]. Hence, Animoto only payed for 4000 virtual instances at peak time and when the peak disappeared, the unused resources were released. Clearly, *elasticity* and *cost-effectiveness* are two of the key features that ensure cloud computing will appeal to more customers.

Nevertheless, true elasticity and cost-effectiveness in the pay-per-use cloud business model have not yet been achieved [3] [4]. The problems over- and under-provisioning resources are still present in cloud service usage [5]. It is not a trivial challenge for cloud web application providers to optimally scale resources up or down. Because web applications have a large number of potential users and web traffic can be highly dynamic, a web application may receive highly bursty requests and become overwhelmed. When a web server becomes overloaded, its users will perceive longer delays or even lose service; meanwhile, even the web server is idle or non-peak periods,a charge is still made for the resource. In addition, complex cost and price models are involved in cloud service usage. For instance, one of the primary price models in Amazon Web Services (AWS) charges by the hour for the number of instances run even if those instances are idle [6]. Therefore, it would be desirable to have an optimal automated and dynamic mechanism for such resource allocation.

To address these challenges, in this paper, we propose a novel scheme for achieving virtual machine (VM) level auto-scaling of cloud resources with optimal cost-latency trade-off for the web application providers. Our proposed scheme strives to allocate just enough resources to applications to minimize resource waste while avoiding service level agreements (SLAs) violations without requiring manual intervention. Three main problems need to be solved to achieve our goal: (1) to predict correctly how many resources are demanded in each time-unit of re-allocation; (2) to adaptively adjust the resource cap based on the predicted resource demands; (3) to design optimization algorithms to make a trade-off decision between cost and latency, while meeting the cost constraints and SLAs on latency metrics.

By leveraging machine learning techniques to analyse the time series history data of web requests, we discover the main features that are primarily used to predict the average number of web requests in a future time-unit (in this paper, we use a unit of one hour). Considering the predicted average value as an expectation of the distribution of web requests which will be allocated to a cloud resource to process in the coming hour, we model the relationship between the number of VM instances and the latency (or response time) by applying a $M/M/m$ model in queueing theory. The true allocated number of VM instances adds padding to predicted resource demands. Taking the cloud price model into account, a multiple optimization model between cost and latency with constraint conditions is developed.

The main innovations of this paper are summarized as follows:
(1) From the web application provider's point of view, to uncover the features of seasonal time patterns by analysing the history data of web application requests, for the purpose of predicting future resource demands;
(2) In VM-level scaling, considering the waiting time of web requests to be executed on VMs as a distribution so that our proposed scheme can be applied to those web applications that have the similar scenarios, rather than being limited to one specific web application;
(3) In each time-unit re-allocation, treat the predicted web requests as a distribution to allocate resources, instead of using an average value, to reduce prediction error and SLA violations;
(4) Proposed a novel resource auto-scaling scheme with prediction and cost-latency trade-off optimization, without manual intervention.

We have implemented the scheme on Amazon AWS and evaluated it by using three real-world web log datasets. The experiment results show that the scheme achieves resource auto-scaling with low prediction errors, as well as optimal resource allocation with scalar cost-latency trade-off and low SLA violations.

The remainder of this paper is organized as follows. In the next section, related works are discussed. Section III describes the modeling of the system including the scheme overview and models for addressing the above four abjectives. In Section IV, the experimental evaluation results are analysed. Finally, our conclusions and future work are given.

II. Related Works

This section summarizes the related state-of-the-art works in the field of cloud resource allocation to achieve automated scaling.

Existing cloud infrastructure as a service (IaaS) providers offer scaling mechanisms to manage their compute resources, such as Amazon AWS AutoScaling [7] and RightScale [8]. These mechanisms require application owners to manually specify scaling rules, and it is hard to achieve full automation. Even the ElastiCache [9] announced recently by Amazon still cannot scale automatically without explicit user's intervention, although users have the option to scale ElasticCache to tailor it to their requirements.

Several research works[10][11][12][13] have extensively studied applying control theory to achieve adaptive fine-grained resource allocations based on feedback of service level objective (SLO) conformity. However, such approaches often have parameters that need to be specified or tuned offline, and some time to converge to optimal decisions is required. [14] proposed a dynamic resource allocation scheme by multiplying estimated resource usage based on different QoS levels. [15] proposed workload prediction using linear regression and histogram based methods. In [16], a web cache model was introduced to adjust IaaS resources to assist application providers in making a trade-off between cost and performance objectives. A lightweight scaling algorithm was proposed by [5] to enable fine-grained scaling application at the level of underlying resources. The authors in [17] and [3] aimed to find a satisfiable balance between reducing energy consumption and operating cost while maintaining a satisfactory service level for minimizing the number of idle VMs. In [18], Gong et al. proposed a lightweight online resource demand prediction scheme that can handle both cyclic and non-cyclic workloads.

Little research has been conducted on behavior analysis for cloud customers' resource demands, as the cloud services have been appeared for couple of years. In the absence of deep understanding behavior of customer resource demands, one possible way of estimating the resource demands is to leverage time series prediction/forecasting techniques to model it from the historical data. Our work attempts to do research work on cloud resource auto-scaling from this view.

III. System Modeling

In this section, we introduce the overview of the proposed scheme and describe the system model used in this paper.

A. Overview of the Scheme

Our scheme scales the cloud resource up or down (or NOP) by time-unit re-allocating based on predicted optimal resource demands. Web application providers can specify their budgetary constraints and SLA in respect of latency for their applications. In each time-unit, web application providers can be notified of the total cost, SLA violations and re-allocation state (e.g. scaling up or down or NOP) by using the optimal resource auto-scaling scheme. Notice that in practical applications, an unpredictable burst of number of requests will happen as a similar situation as the Animoto experienced. To tackle this unpredictable scenario, this scheme monitors the waiting queue of requests to be processed in real-time. Once the length of the queue is bigger than a threshold, the scheme could dynamically append VMs for processing the exceed number of requests. Figure1 illustrates the overview execution paradigm facilitated by the scheme.

As shown in the Figure 1, the main steps of our scheme are outlined as follows.

1) to collect request records as the history data;
2) to analyse the history data hourly and predict the number of requests for the next time-unit (Subsection III-B);
3) to discover the optimal number of VMs by utilizing the Optimization Model (Subsection III-C);

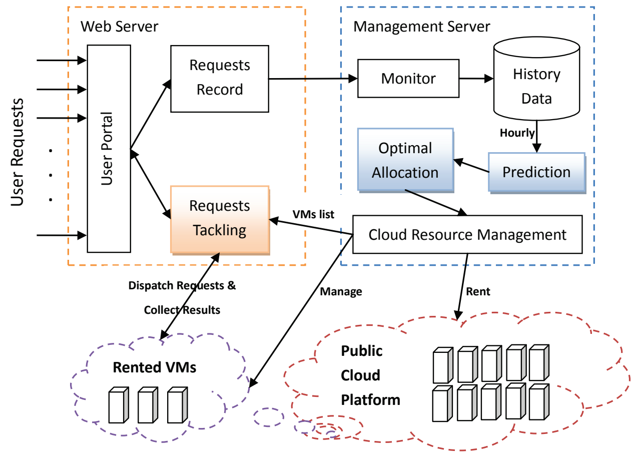

Figure 1. Optimal Cloud Resource Auto-scaling Overview.

4) to scale the resource(VMs) up or down or NOP from public cloud platform

B. Prediction Model

Because launching a VM instance takes several tens of seconds to minutes, we propose a predictive-driven resource scaling approach. As the parameters of a VM underlying resource, such as CPU, memory, I/O or network bandwidth, are not necessarily dependent [19], it is not trivial to model resource demand prediction directly within these parameters (i.e. resource-level). In our work, we predict the web request distribution in each time-unit. Subsequently, we model the resource demand based on the predicted web request distribution at a VM-level (that is, by considering the number of VMs, rather than the number of configuration parameters as the resource demand quantity).

1) Definition: To predict the number of web requests, the web request data is generally denoted as a time series [20]:$\{X(t); t \in T\}$, where T is an index of the time fragment, and $X(t)$ is the random variable, representing the total number of requests that arrive in the t time fragment. The prediction problem can be defined as follows: given the current and past observed values $(X(t-k), X(t-k+1), ...X(t-1), X(t))$, predict the future value $X(t+p)$, where k is the length of the history data used for prediction and p is the predictive time fragment.

2) Key Features Identification: There are several different techniques for predicting the future behavior of a time series, given present and past values, such as time-domain and frequency-domain; model-based and non-model-based; linear and nonlinear techniques [21][22]. Considering most online web requests have a seasonal or periodical behavior to some extent, we design a novel Linear Regression approach for prediction by using an autocorrelation function to identify the key features.

For instance, a mail server usally experiences the highest web traffic volume every Monday morning, while at midnight, web requests drop to a low level; also, the number of requests will be much higher on weekdays than on weekend. Many more users may request a mail service on festivals and holidays than on other days. Therefore, the web requests behavior pattern can be established and key features such as hourly,daily, weekly,monthly, seasonally, etc., can be identified by analysing the history data.

Considering a web requests time series $(X(t-k), X(t-k+1), ...X(t-1), X(t))$, where $X(t)$ represents the total requests to arrive during this time fragment t, the web request in the next time fragment t is related to the web request volume in prevous time fragments whether there are two, ten, or hundreds of time fragments. We utilize a linear model to present their relationship.

$$X(t) = \sum_{i=1}^{N} w_i X(t-i) \tag{1}$$

where w_i is the weight of different related $X(t-i)$, N is the number of related time fragments.

Based on equation (1), we can estimate all w_i by utilizing a linear regression method to obtain the prediction model. If the related time fragments are too many, an overfitting problem will occur and prediction accuracy may be reduced. Thus the top key features which mainly determine the predicted value should be identified. In this work, we apply the autocorrelation function to identify the key features [22]. For the request state in each time fragment $X(t)$, its autocorrelation with another request $X(t-i)$ is calculated by

$$\rho_{t,t-i} = \frac{E\{[X(t)-\mu][X(t-i)-\mu]\}}{\sigma(t)\sigma(t-i)} \tag{2}$$

For different i, we obtain a vector $V = \{\underset{t}{mean}(\rho_{t,t-i}) | i \in [1, N]\}$. We select K elements with top values from the sorted vector V as the K key correlated features. These selected elements are composed of a new vector $\bar{N}$. For $t' \in [1, \bar{N}]$, the linear regression model can be estimated as follows:

$$X(t) = \sum_{t'=1}^{K} w_{t'} X(t') \tag{3}$$

Figure 2 shows the autocorrelations for lags 0-50. Note that it is characterized by pseudo-periodic behavior with a cycle-length of about 12 time points.

3) Modelling the Relationship between Cost and Latency: To estimate the relationship between web request volume, cost and latency, we take the following into consideration in our scheme design: (1) cost (C) prediction depends on the number (M) of VMs changing, e.g C=$f(m)$; (2) latency (L) consists of execution time (T_s) and waiting time for executing (T_q); (3) the arrivals of requests to be processed

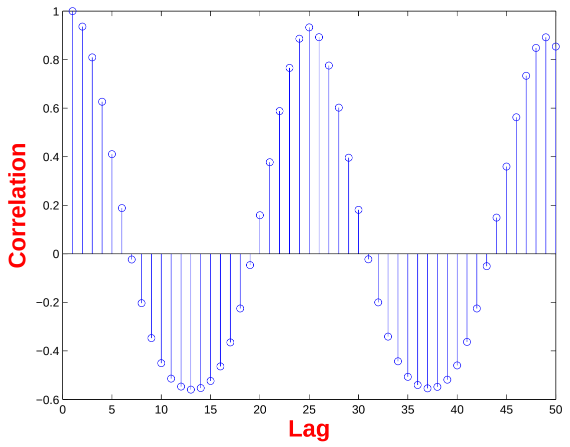

Figure 2. Autocorrelation

on VMs obey a Poisson distribution with rate λ , and the executed requests on VMs are also considered as a Poission distribution with rate μ. For convenience, each web server is installed on one VM and all VMs belong to the same type of instance with the same process capacity. These allocated Vms come from an infinite cloud-based resource pool.

We employ the queueing theory technique to model this relationship and we consider the arrival-execution of requests on VMs as a birth-death process which is a special Markov chain [23], as shown in Figure 3.

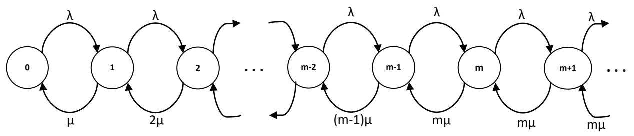

Figure 3. Rate Transition Rate for the Web Requests Process on VMs

Due to the allocation of multiple servers (or VMs) in the scheme, this process of the arrival-execution of requests on VMs is modeled as $M/M/m$ queueing: the arrivals are poisson with rate λ ($\lambda = E(X(t))$, and each VM has an independent and identical distribution exponential execution-time distribution with mean μ . Since the execution-time of the web request on a given type of VM can be obtained by experiments, the execution-time T_s is known and $\mu = 1/E(T_s)$.

There are m VMs in the system and we allocate one web request to be processed on each one VM in parallel at every instant time-point with a constant rate, so that the "birth" rate is $\lambda_n = \lambda$ for all n. On the other hand, the rate of request completions (or "deaths") depends on the number of VMs in the system. If there are m or more requests in the system, then all m servers must be busy at the instant time-point. Since each VM processes requests with rate μ, the combined process-completion rate for the system is $(m\mu)$. When there are fewer than m customers in the system, e.g. $i < m$, only i of the m VMs are busy and the combined service-completion rate for the system is $(i\mu)$. Hence μ_i may be written as

$$\mu_i = \begin{cases} i\mu & 1 \leqslant i < m, \\ m\mu & i \geqslant m \end{cases} \tag{4}$$

Based on the Markov chain in Figure 3, we can obtain the steady-state probabilities p_i

$$p_i = \begin{cases} \frac{\lambda^i}{i!\mu^i} p_0 & 1 \leqslant i < m, \\ \frac{\lambda^i}{m^{(n-m)} m! \mu^i} p_0 & i \geqslant m \end{cases} \tag{5}$$

To obtain the value of p_0, we use the condition that the probabilities must sum to 1 ($\sum_{i=0}^{\infty} p_i = 1$).

$$p_0 = (\sum_{i=0}^{m-1} \frac{\lambda^i}{i!\mu^i} + \sum_{i=m}^{\infty} \frac{\lambda^i}{m^{(i-m)} m! \mu^i})^{-1} \tag{6}$$

With the steady-state probabilities p_i, we can calculate the expected queue size L_q. L_q equals zero when the request number i is no more than VM number m, and is equal to $(i - m)$ when the request number i is more than the VM number n , and thus,

$$L_q = \sum_{i=m+1}^{\infty} (i - m) p_i \tag{7}$$

Based on Little's Formula [23] $L_q = \lambda T_q$, where $T_q = E(t_q)$ is the expected length of the waiting time in queue t_q.

$$T_q = \frac{L_q}{\lambda} = p_0 (\frac{(\lambda/\mu)^m}{m!(mu)(1 - (\frac{\lambda}{m\mu})^2}) \tag{8}$$

With the expected waiting time T_q, we can calculate the expected response time (average latency) L by the equation below:

$$L(\lambda, \mu, m) = T_q + T_s \tag{9}$$

C. Optimization Model

1) Objective Function: Recall that the web application provider's greatest concern is to maximize profit (e.g. by minimizing cost) while providing high quality service (e.g. by minimizing latency) with lower SLA violation. However, these two factors are in conflict. As in our cloud-based web system, with the cost demand on the number of allocated VMs, we can reduce the number of VMs to keep the cost as low as possible when there are insufficient VMs to process requests, but the waiting time in the queue will be too long. To solve this problem, we exploit the cost-latency trade-off optimization objective function, as follows:

$$arg \min_{m,\lambda,\mu} \Gamma(\lambda, \mu, m) = \alpha * f(m) + (1 - \alpha) * L(\lambda, \mu, m) \tag{10}$$

where $\alpha \in [0, 1]$ reflects the importance ratio of cost and latency.

Due to the different scale of the number of VMs and latency, we can normalize the latency by equation

$$G = L/T \tag{11}$$

where T is the latency threshold which is defined in SLAs. To normalize the number of VMs, we consider the equation

$$C' = F(m) = \frac{f(m) - f_{max}(m)}{f_{max}(m) - f_{min}(m)} \tag{12}$$

where $f(m)$, $f_{max}(m)$ and $f_{min}(m)$ refer to the VMs cost per time-unit, the least possible cost per time-unit and the maximum possible cost per time-unit,respectively. Then, we derive the following objective function for the optimization.

$$arg \min_{m,\lambda,\mu} \Gamma(\lambda, \mu, m) = \alpha * F(m) + (1 - \alpha) * G(\lambda, \mu, m) \tag{13}$$

Based on predicted requests in unit time t ,λ and μ are given, and the latency function in unit time t can be written as

$$L_t(\lambda, \mu, m) = L_t(m) \tag{14}$$

Considering the need to satisfy web application providers' cost constraints and SLA violation in respect of latency, the final objective function of the cost-latency trade-off in unit time t is obtained by the following:

$$arg \min_{m} \Gamma_t(m) = \alpha * F_t(m) + (1 - \alpha) * G_t(m) \tag{15}$$

subject to:

$$\forall_t : C_t(m) \le C_t max; Pr\{L_t(m) > T\} \leqslant K\% \tag{16}$$

where the SLAs violations constraint K is usually defined as $K \in [2, 5]$ for web applications.

Assuming each server could tackle k requests within time T, the m VMs could tackle mk requests. This means that the SLA will be satisfied when the queue length is less than mk, because all requests could be tackled within time T. By referring to Figure 3, we know that only the previous mk steady-state can satisfy the SLA, and others will violate the SLA. So the equation of SLAs violations constraint can be written by

$$Pr\{L_t(m) \leqslant T\} = \sum_{i=0}^{mk} p_i > (1 - K\%) \tag{17}$$

Algorithm 1 Computing optimal number of VMs

```
input
  $\lambda$ - arrival rate, $\mu$ - process rate per VMs,
  $\alpha$ - priority of cost, $\mathcal{K}$ - threshold of SLA violation
output
  $m$ - optimal number of VMs
1: $minV = \infty$
2: for (n = 1..N) do
3:     $l_t = L(\lambda, \mu, n)$; //Equation (9)
4:     $l_t{}' = normalize(l_t)$; //Equation (11)
5:     $n' = normalize(n)$; //Equation (12)
6:     $newV = \alpha * n' + (1 - \alpha) * l_t{}'$; //Equation (15)
7:     if ( ($n$ satisfy constraint($\mathcal{K}$)) // Equation (17)
       && ($newV < minV$) ) then
8:         $m = n$;
9:         $minV = newV$;
10:    end if
11: end for
```

2) Solving the Optimization Problem: To minimize the objective function, we wish to find an optimal number of VMs m to obtain the cost-latency trade-off values, satisfying all constraints. Clearly, equation 15 is a complex nonlinear function and hard to simplify by mathematical methods. Considering the number of VMs the web application provider purchased is limited, we exploit an exhaustive search algorithm to calculate the Γ with different m, and to find the lowest $Cost$ and the related m, as shown in algorithm 1.

IV. Experimental Evaluation

In this section, we evaluate the performance of the proposed scheme. We first describe experiment setup and datasets used in our experiments, followed by the analysis and discussion of the evaluation results.

A. Experiment Setup and Datasets

We evaluate the performance of the scheme by using three kinds of real-world datasets. We consider the well-known AOL [1] and Sogou [2] search log dataset, as well as another real-world dataset collected by the UTS (University of Technology, Sydney) library, to evaluate the performance. Because most VMs instances in public clouds are charged hourly, the time-unit of re-allocation in our work is the hour-unit. Therefore, we set the length of time fragment as one hour, and aggregate the number of requests for each hour.

We organize the experiment by steps as follows:

1) investigate how the seasonal characters affect the selection of features for prediction modeling (Subsection IV-B1);

[1]http://www.infochimps.com/datasets/aol-search-data
[2]http://www.sogou.com/labs/dl/q-e.html

2) evaluate the prediction model through three datasets (Subsection IV-B2);
3) visualize the performance of the prediction model (Subsection IV-B3);
4) evaluate the allocation performance for the given number of requests (Subsection IV-B4);
5) compare our scheme with other approaches (Subsection IV-B5).

B. Evaluation and Results

1) Features Selection Evaluation: To measure the seasonal characters, we compare the difference between two time periods. We represent the number of requests in each hour as a vector $<v_1, ..., v_i, ...v_{60}>$, where v_i is the requests volume within one minute. We consider each vector as a distribution, and apply the Kullback-Leibler (KL) divergence to measure the difference between two distribution probabilities.

$$D_{KL}(P||Q) = \sum_i \ln(\frac{P(i)}{Q(i)})P(i) \quad (18)$$

Because the KL divergence is a non-symmetric measure, we utilize a variant Symmetrizing KL (SKL) divergence [24] to evalute as

$$SD_{KL}(P||Q) = \frac{D_{KL}(P||Q) + D_{KL}(Q||P)}{2} \quad (19)$$

By taking the hourly number of requests as an element, the requests volume in a day can be considered as a 24-length vector. Each vector is treated as a distribution, and can be calculated the SKL divergence with another hourly vector. Similarly, the hourly vector can be extended to a weekly or monthly vector. Table I shows the average SKL divergences on hourly, daily and weekly vectors with three datasets.

Table I
AVERAGE SKL DIVERGENCE ON DIFFERENT PERIOD VECTOR

Dataset	Hour	Day	Week
AOL	0.0324	0.0283	0.0229
UTSlib	0.0667	0.0302	0.0577
Sogou	0.0054	0.0086	0.0110

For the SKL divergence, 1 presents the greatest distance and 0 describes the smallest distance. All the SKL divergences in Table I are small which demonstrates the three datasets have highly seasonal characters and the number of requests can be predicted by using the history data.

Before learning the prediction model, we need to select the key features for the linear regression model. Table II reprents the top 10 correlated features. The results in Table II show that the request volume in time-unit t is most correlated to that of the first previous unit-time $t-1$.

Table II
TOP 10 CORRELATED LAGS

Period	Correlated lag (ordered by correlation descent)
AOL	1,2,3,4,5,145,144,146,6,143
UTSlib	1,2,169,25,168,24,170,145,26,3
Sogou	1,25,2,49,24,26,73,48,50,97

2) Evaluation Methods: We choose several common measurements for the regression model, such as Root Mean Squared Error (RMSE), Relative Squared Error (RSE), Mean Absolute Error (MAE), Relative Absolute Error (RAE), and coefficient of determination (R^2).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(p_i - a_i)^2}{n}} \quad (20)$$

$$RSE = \frac{\sum_{i=1}^{n}(p_i - a_i)^2}{\sum_{i=1}^{n}(\bar{a_i} - a_i)^2} \quad (21)$$

$$MAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{n} \quad (22)$$

$$RAE = \frac{\sum_{i=1}^{n}|p_i - a_i|}{\sum_{i=1}^{n}|\bar{a_i} - a_i|} \quad (23)$$

$$R^2 = \frac{\sum_{i=1}^{n}(p_i - \bar{p_i})^2}{\sum_{i=1}^{n}(a_i - \bar{a_i})^2} \quad (24)$$

where a is the actual value, p is the predicted value.

We choose 10-fold cross validation as the evaluation method. The Table III shows the performance of the regression model on three datasets.

Table III
PERFORMANCE OF REGRESSION MODEL

Data	Avg Req	RMSE	RSE	MAE	RAE	R^2
AOL	1.6*10^3	191	0.05	140	0.19	0.98
UTSlib	2.9*10^4	4582	0.10	3082	0.25	0.96
Sogou	7.6*10^4	5617	0.02	3555	0.10	0.99

3) Prediction Model Evaluation: In a practical application, a padding is added to the predicted value as the cap (U) of prediction.

$$U = (1 + padding) * prediction \quad (25)$$

We evaluate the prediction accuracy by utilizing the confidence interval $Pr(x < U)$, which represents the probability that real demands (x) are less than the cap (U) of the prediction. To select a good padding value, we measure the relationship between the padding value and the confidence interval, as shown in Table IV.

The Figure 4 shows that our scheme achieves the good prediction on both number of requests and resource demands, and that the pading value can be dynamically adjusted well in each time interval.

Table IV
THE CONFIDENCE INTERVAL WITH DIFFERENT PADDINGS

Padding (%)	Confidence Interval Pr{x≤U}		
	AOL (%)	UTSlib (%)	Sogou (%)
5	69.27	68.63	86.84
10	82.96	81.16	95.44
15	91.46	89.67	98.60
20	95.49	94.43	99.30
25	97.57	97.10	99.30
30	98.50	98.17	99.65
35	99.13	99.05	100
40	99.42	99.46	100
45	99.56	99.64	100
50	99.76	99.79	100

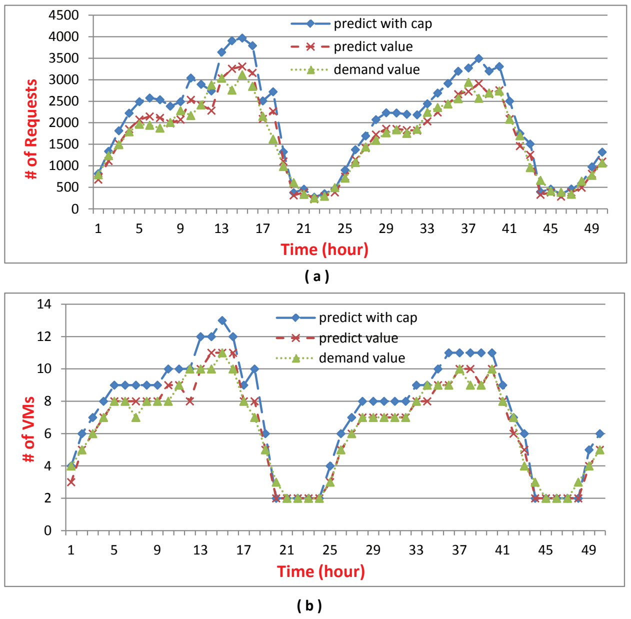

Figure 4. Prediction and Allocation with padding.

4) *Allocation Evaluation:* Our allocation approach is related to the arrival rate λ (per minute), process rate μ (per minute), maximal process time T (s), SLA violation ratio threshold K, and cost priority α. We define $r = \lambda/\mu$ as the minimal required number of VMs. m is the optimal number of VMs allocated by our scheme.

With the given $\mu = 10$, $T = 60$, $K = 2\%$ and $\alpha = 0.8$, we change the λ from 10 to 300. Figure 5 (a) shows that a bigger padding $(m-r)$ should be allocated when the number of requests increases. Meanwhile the Figure 5 (b) shows that the relative ratio between m and r (i.e.m/r) decreases to be closed to 1, which means the scheme achieves a good cost-effectiveness, when the number of requests rises.

5) *Performance Evaluation for a Web Application:* We implement our scheme on Amazon AWS with a Web application. This scheme can rent or lease VM instances automatically from Amazon EC2. To simplify the problem, our experiment only considers the cost of VMs with same type of instances. We simulate the frequency of requests based on the real datasets, and the process time of requests obeys Poisson distribution ($\frac{1}{\mu} = 6$ seconds).

We compare our approach (QT) with other three approaches: Peak, PEAK($\times 3/4$) and Cap($\times 2$). For Peak approach, the number of VMs is always allocated based on the peak value. While PEAK($\times 3/4$) is an approach to reduce cost by allocating the number of VMs as 3/4 of peak value. For CAP($\times 2$) approach, the resource cap is set as the 2 times of the minimal number of VMs which can satisfy the predicted the number of requests by considering all request arrived with average rate.

As Table V shows, our approach allocates less resource while achieves better performance comparing to the PEAK($\times 3/4$) and Cap($\times 2$). Comparing to the PEAK, our approach reduces much less numbers of VMs, although with a little bit higher SLA violation rate.

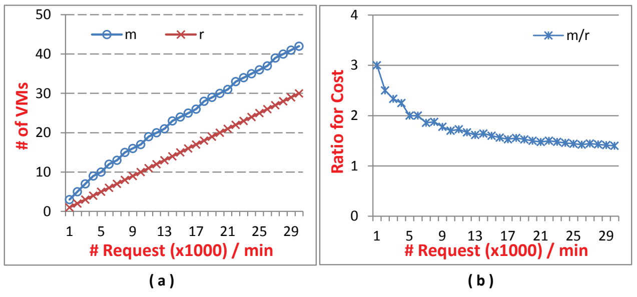

Figure 5. Allocation with Queuing Theory.

Table V
COMPARE THE PREDICTION ACCURACY FOR DIFFERENT METHOD.

Dataset	Approach	# Req /h	# VMs /h (avg)	Violate (%)	Avg Tq /h (s)
AOL	PEAK	1916	13.25	0.03	6.15
	PEAK (×3/4)	1916	9.94	0.63	16.15
	CAP(×2)	1916	7.33	0.57	15.61
	QT	1916	**7.21**	**0.18**	**9.96**
UTSlib	PEAK	2165	21.00	0.02	7.46
	PEAK (×3/4)	2165	15.75	0.45	19.15
	CAP(×2)	2165	8.25	0.24	13.73
	QT	2165	**7.75**	**0.20**	**11.87**
Sogou	PEAK	2954	25.67	0.06	8.32
	PEAK (×3/4)	2954	19.23	1.02	26.15
	CAP(×2)	2954	10.67	0.76	18.35
	QT	2954	**9.70**	**0.54**	**13.54**

V. CONCLUSION & FUTURE WORKS

In this paper, we proposed an optimal VM-level auto-scaling scheme with cost-latency trade-off. In each re-allocation time-unit, we predicted the number of requests based on history data by exploiting machine learning techniques and time series analytics. Considering the predicted results, we discovered an optimal number of VMs by utilizing queueing thoery and multi-objective optimizaiton. Based on the optimal VMs demanded to be allocated, the system make a decision of scaling up or scaling down or NOP. The experimental results demonstrate that the proposed scheme can balance the cost and desired latency. Compare with other methods, our schme presents superior price-performance ratio across three real-world datasets. This research can potentially accelerate the migration of web applications to cloud service. The consideration of more general queueing models and other types of VMs (e.g multi-tenant shared) to extend this work can be conducted in the future.

Acknowledgments

The work presented in this paper was supported by the Australian Research Council (ARC) under discovery grant DP110103733. Part of experimental datasets are provided by UTS library.

References

[1] M. Armbrust, A. Fox, R. Griffith *et al.*, “Above the clouds: A berkeley view of cloud computing,” *University of California, Berkeley, Tech. Rep.*, vol. 28, 2009.

[2] “Animoto case in rightscale blog,” http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/.

[3] B. Dougherty, J. White, and D. Schmidt, “Model-driven auto-scaling of green cloud computing infrastructure,” *Future Generation Computer Systems*, vol. 28, no. 2, pp. 371–378, 2012.

[4] M. Stillwell, D. Schanzenbach, F. Vivien, and H. Casanova, “Resource allocation algorithms for virtualized service hosting platforms,” *Journal of Parallel and Distributed Computing*, vol. 70, no. 9, pp. 962–974, 2010.

[5] R. Han, L. Guo, M. Ghanem, and Y. Guo, “Lightweight resource scaling for cloud applications,” in *Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on*, 2012, pp. 644–651.

[6] “Amazon ec2 pricing,” http://aws.amazon.com/ec2/pricing/.

[7] “Amazon auto scaling,” http://aws.amazon.com/autoscaling/.

[8] “Rightscale,” http://www.rightscale.com/.

[9] “Amazon elasticache,” http://aws.amazon.com/elasticache/.

[10] P. Padala, K. Hou, and K. e. a. Shin, “Automated control of multiple virtualized resources,” in *Proceedings of the 4th ACM European conference on Computer systems*, 2009, pp. 13–26.

[11] H. Lim, S. Babu, and J. Chase, “Automated control for elastic storage,” in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 1–10.

[12] A. Sangpetch, A. Turner, and H. Kim, “How to tame your vms: an automated control system for virtualized services,” in *Proceedings of the 24th international conference on Large installation system administration*, 2010, pp. 1–16.

[13] E. Kalyvianaki, T. Charalambous, and S. Hand, “Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters,” in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 117–126.

[14] H. Goudarzi, M. Ghasemazar, and M. Pedram, “Sla-based optimization of power and migration cost in cloud computing,” in *Cluster, Cloud and Grid Computing (CCGrid), 12th IEEE/ACM International Symposium on*, 2012, pp. 172–179.

[15] S. Islam, J. Keung, K. Lee, and A. Liu, “Empirical prediction models for adaptive resource provisioning in the cloud,” *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[16] F. Kabir and D. Chiu, “Reconciling cost and performance objectives for elastic web caches,” in *Proceedings of the 2012 IEEE International Conference on Cloud and Services Computing (CSC'12)*, 2012.

[17] Y. Jiang, C. Perng, T. Li, and R. Chang, “Self-adaptive cloud capacity planning,” in *Services Computing (SCC), 2012 IEEE Ninth International Conference on*, 2012, pp. 73–80.

[18] Z. Gong, X. Gu, and J. Wilkes, “Press: Predictive elastic resource scaling for cloud systems,” in *Network and Service Management (CNSM), 2010 International Conference on*, 2010, pp. 9–16.

[19] C. Xu, J. Rao, and X. Bu, “Url: A unified reinforcement learning approach for autonomic cloud management,” *Journal of Parallel and Distributed Computing*, 2011.

[20] W. Zhao and H. Schulzrinne, “Predicting the upper bound of web traffic volume using a multiple time scale approach,” in *Proceedings of International World Wide Web Conference (WWW)*, p. 251.

[21] C. Chatfield, *Time-series forecasting*. Chapman & Hall/CRC, 2000.

[22] W. Woodward, H. Gray, and A. Elliot, *Applied Time Series Analysis*, ser. Statistics: a Series of Textbooks and Monographs, 2011.

[23] D. Gross, J. Shortle, J. Thompson, and C. Harris, *Fundamentals of queueing theory*. Wiley-Interscience, 2011, vol. 627.

[24] D. Johnson, S. Sinanovic *et al.*, “Symmetrizing the kullback-leibler distance,” *IEEE Transactions on Information Theory*, vol. 1, no. 1, pp. 1–10, 2001.