2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops

Real-Time Performance Prediction for Cloud Components

Yilei Zhang[†], Zibin Zheng[†] and Michael R. Lyu[†§]
[†]*Department of Computer Science and Engineering*
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong
{ylzhang, zbzheng, lyu}@cse.cuhk.edu.hk
[§]*School of Computer Science*
National University of Defence Technology
Changsha, China

***Abstract*—Cloud computing provides access to large pools of distributed components for building high-quality applications. User-side performance of cloud components highly depends on the remote server status as well as the unpredictability of the Internet, which are variable over time. It is an important task to explore an method to predict the real-time performance of cloud components. To address this critical challenge, this paper proposes a prediction framework to predict real-time component performance effectively. Our prediction framework builds feature models based on the past usage experience of different users and employs time series analysis techniques on feature trends to make performance prediction. The results of large-scale experiments show the effectiveness and efficiency of our method.**

I. Introduction

Cloud computing [1] is a new type of Internet-based computing, whereby shared resources, software, and information are provided to computers and other devices on demand. A cloud application typically consists of multiple cloud components communicating with each other over application programming interfaces, usually Web services [2]. How to build high-quality cloud applications becomes an urgent and crucial research problem.

Low response-time is one of the most important requirements of cloud applications. However, the response-time performance of cloud applications is greatly influenced by the invoked cloud components. Typically, the cloud components are deployed in different geographical locations and invoked via Internet connections. Moreover, the remote cloud components may be deployed on cheap and poor performing servers, leading to a decrease of performance. It becomes a great challenge to build cloud applications with good response-time performance.

Usually, cloud application customers expect to receive a certain level application performance as specified in Service Level Agreement (SLA). The customers expected performance of cloud applications should be guaranteed at run-time. In order to maintain the performance of cloud applications, which are typically running in highly dynamic environments, real-time performance of the involved cloud components needs to be continuously monitored. Moreover, real-time performance information of cloud components is important for improving the overall performance of the cloud applications by replacing poor performing components with better ones.

Based on the above analysis, providing real-time performance information of cloud component is essential for cloud application designers to build high-quality applications and to maintain the performance of the systems at run-time. However, evaluating the real-time performance of cloud applications is not an easy task, due to:

- Executing invocations for evaluation purposes becomes too expensive, since cloud providers who maintain and host cloud components (e.g., Amazon EC2[1], Amazon S3[2], etc.) may charge for invocations.
- More cloud components are available over the Internet, conducting performance evaluations on all cloud components becomes time-consuming and impractical.
- Performance of cloud components is highly related to the time of invocation, since the server status (e.g., workload, number of clients, etc.) and the network environment (e.g., congestions, etc.) may change over time. Real-time performance testing may introduce extra transaction workload, which may impact the user experience of using the systems. Moreover, with introduced transaction workloads, the performance evaluating may not be accurate.

It becomes an urgent task to explore a personalized prediction approach for efficiently estimating the real-time performance of cloud applications for different users. In this paper, we propose a system performance estimation framework for providing personalized performance information to the customers at run-time. We collect time-aware performance information from geographically distributed component users. We then extract the features of users and components in each time slice. By analyzing the trend of the feature changes, we estimate the features of users and components in the coming future. Then the personalized

[1]http://aws.amazon.com/ec2
[2]http://aws.amazon.com/s3

978-0-7695-4669-8/12 $26.00 © 2012 IEEE
DOI 10.1109/ISORCW.2012.29

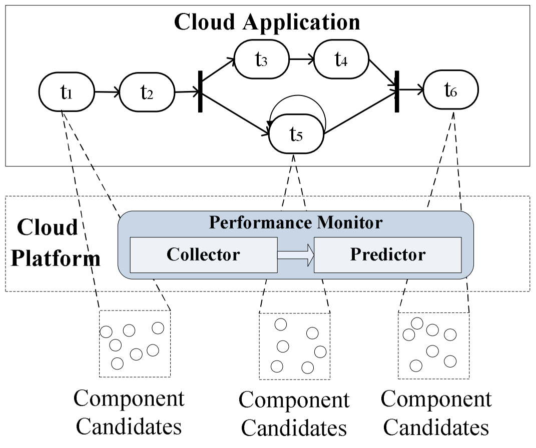


Figure 1. Cloud Application Architecture

performance of cloud components is predicted by evaluating how the features of users apply to features of components.

In summary, this paper makes the following contributions:

- We propose an real-time performance prediction framework for estimating the user observed performance of cloud components. Our approach employs the past usage experiences of different users to efficiently predict the performance of cloud components.
- We conduct large-scale extensive experiments for evaluating the performance of our proposed approach.

The rest of this paper is organized as follows: Section II describes the cloud application architecture and introduces the real-time performance prediction procedures. Section III presents our real-time cloud component performance prediction approach in detail. Section IV presents the experimental results, Section V discusses related work and Section VI concludes the paper.

II. System Architecture

Figure 1 shows a typical cloud application architecture. Within a cloud application, the complicated functions can be implemented by combing several abstract tasks. For each abstract task, an optimal cloud component is selected from a set of functionally equivalent component candidates. By composing the selected components, a cloud application instance is implemented for task execution. Typically the cloud component candidates are distributed in different geographical locations and time zones. When invoked through communication links, the user-side usage experiences are influenced by the network environments and the server-side status at invocation time. Since cloud applications are increasingly running on large numbers of dynamic components, users often encounter highly dynamic and uncertain performance of cloud applications.

As shown in Figure 2, the real-time performance prediction mechanism proposed in this paper contains three phases.

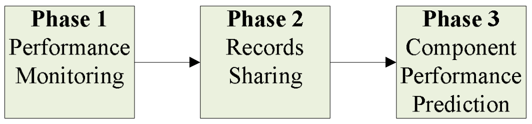


Figure 2. Real-Time Performance Prediction Procedures

In phase 1, each component user keeps local performance records of the cloud components. In phase 2, local cloud component usage experiences are shared among different component users. Each user is encouraged to contribute its local records to obtain records from other users. By contributing more individually observed cloud component performance information, a component user can obtain more global performance information from other users, thus obtaining more accurate cloud component performance prediction values. By combining the local records of component performance, global performance information for all component is obtained. In phase 3, time series analysis [3] is conducted on the extracted time-specific user features and component features. A performance model is built for personalized real-time component performance prediction. The detailed real-time performance prediction approach is presented in Section III.

In this paper, we focus on the design of *Performance Monitor* within a cloud platform. User-side real-time performance of cloud components is monitored by the module *Performance Monitor*. The *Performance Monitor* consists of two sub-units: *Collector*, which is used to collect performance information from various component users and *Predictor*, which is supposed to provide real-time performance prediction for different component users.

III. Real-Time Cloud Component Performance Prediction

In this section, we propose a collaborative method to make personalized real-time performance prediction of cloud components for different users. We first propose a latent feature learning algorithm to build the time-aware user-specific and component-specific feature models in Section III-A. The performance of components is then predicted by applying the proposed prediction algorithm in Section III-B.

A. Time-Aware Latent Feature Model

As shown in Fig. 3, let U be the set of m users and C be the set of n cloud components. In each time slice t, the observed response-time from all users is represented as a matrix $R(t) \in \mathbb{R}^{m\times n}$ with each existing entry $r_{ui}(t)$ representing the response-time of component i observed by user u in time slice t. Given the set of matrices $\Psi = \{R(k)|k < t_c\}$, matrix $R(t_c)$ should be predicted representing the expected response-time of components in time slice t_c.

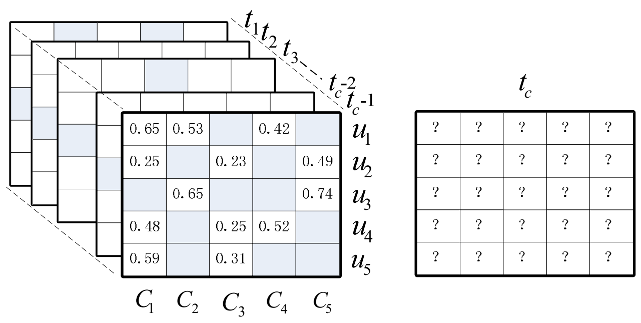


Figure 3. A Toy Example of Performance Prediction

Without loss of generality, we can map the response-time values to the interval $[0,1]$ using the following function:

$$f(x) = \begin{cases} 0, & \text{if } x < r_{min} \\ 1, & \text{if } x > r_{max} \\ \frac{x - r_{min}}{r_{max} - r_{min}}, & \text{otherwise} \end{cases}$$

where r_{max} and r_{min} are the upper bound and lower bound of the response-time values, respectively, which can be defined by users.

In order to learn the latent features of users and components, we employ a matrix factorization technique to fit a feature model to user-component matrix in each time slice. The factorized user-specific and component-specific features are utilized to make further performance prediction. The idea behind the feature model is to derive a high-quality low-dimensional feature representation of users and components by analyzing the user-component matrices. It is noted that there is only a small number of features influencing performance experiences, and that a user's performance experience vector is determined by how each feature is applied to that user and the corresponding component. Examples of physical features are network distance between the user and the server, the workload of the server, etc. Latent features are orthogonal representation of the decomposed results of physical features. Consider the matrix $R(t) \in \mathbb{R}^{m \times n}$ consisting of m users and n components. Let $p(t) \in \mathbb{R}^{l \times m}$ and $q(t) \in \mathbb{R}^{l \times n}$ be the latent user and component feature matrices in time slice t. Each column in $p(t)$ represents the l-dimensional user-specific latent feature vector of a user and each column in $q(t)$ represents the l-dimensional component-specific latent feature vector of a component. We employ an approximating matrix to fit the user-component matrix $R(t)$, in which each entry is approximated as:

$$\hat{r}_{ui}(t) = p_u^T(t) q_i(t) \tag{1}$$

where l is the rank of the factorization which is generally chosen so that $(m+n)l < mn$, since $p(t)$ and $q(t)$ are low-rank feature representations [4].

We construct a cost function to evaluate the quality of approximation in each slice. The cost function is usually defined by evaluting the distance between two non-negative matrices. In this paper, due to the reason that there are a large number of missing values in practice, we only factorize the observed entries in matrix $R(t)$.

$$\begin{aligned} & \min \mathcal{L}(p_u(t), p_i(t)) \\ = \quad & \frac{1}{2} \sum_{u=1}^{m} \sum_{i=1}^{n} I_{ui} (r_{ui}(t) - g(\hat{r}_{ui}(t)))^2 \\ + \quad & \frac{\lambda_1}{2} ||p(t)||^2 + \frac{\lambda_2}{2} ||q(t)||^2, \end{aligned} \tag{2}$$

where $\lambda_1, \lambda_2 > 0$, I_{ui} is the indicator function that is equal to 1 if user u invoked component i during the time slice t and equal to 0 otherwise. To avoid the overfitting problem, we add two regularization terms to Eq. (2) to constrain the norms of $p(t)$ and $q(t)$ where $||\cdot||^2$ denotes the Frobenius norm. The optimization problem in Eq. (2) minimizes the sum-of-squared-errors objective function with quadratic regularization terms. $g(x) = 1/(1 + exp(-x))$, which maps $\hat{r}_{ui}(t)$ to the interval $[0,1]$.

A local minimum of the objective function given by Eq. (2) can be found by performing incremental gradient descent in feature vectors $p(t)$ and $q(t)$:

$$\begin{aligned} \frac{\partial L}{p_u(t)} = \quad & I_{ui}(g(\hat{r}_{ui}(t)) - r_{ui}(t)) g'(\hat{r}_{ui}(t)) q_i(t) \\ & + \lambda_1 p_u(t), \end{aligned} \tag{3}$$

$$\begin{aligned} \frac{\partial L}{q_i(t)} = \quad & I_{ui}(g(\hat{r}_{ui}(t)) - r_{ui}(t)) g'(\hat{r}_{ui}(t)) p_u(t) \\ & + \lambda_2 q_i(t). \end{aligned} \tag{4}$$

B. Real-Time Performance Prediction

Given the latent feature vectors of users and components in time slices before t_c, the latent feature vectors in time slice t_c can be predicted by precisely modeling the trends of features. Intuitively, older features are less correlated with a component's current status or a user's current characteristics. To characterize the latent features at time slice t_c, the prediction calculation should rely more on the information collected in the latest time slices than that collected in older time slices. In order to integrate the information from different time slices, we therefore employ the following temporal relevance function:

$$f(k) = e^{-\alpha k}, \tag{5}$$

where k is the amount of time that has passed since the corresponding information was collected. $f(k)$ measures the relevance of information collected from different time slices for making prediction on latent features at time t_c. Note that $f(k)$ decreases with k. By employing the temporal relevance function $f(k)$, we can assign a weight for each latent feature vector depending on the collecting time when making the prediction. In the temporal relevance function, α controls the decaying rate. By setting α to 0, the evolutionary nature of the information is ignored. A constant temporal relevance value of 1 is assigned to latent feature vectors in all the time slices, which means latent feature vectors in time slice t_c are predicted simply by averaging the vectors before time slice t_c. Since $e^{-\alpha}$ is a constant value, the value of temporal relevance function can be recursively computed: $f(k+1) = e^{-\alpha} f(k)$.

By analyzing the collected performance data, we obtain two important observations: (1) Within a relatively long time period such as one day or one week, the component performance observed by a user may vary significantly due to the highly dynamic component side status (e.g., workloads of storage component may increase sharply at the opening of stock markets.) and user side environment (e.g., network latency would increase during the office hours). (2) Within a relatively short time period such as one minute or one hour, a component performance observed by a user is relatively stable. The above two observations indicate that the feature information of latent feature vectors in time slice t_c can be predicted by utilizing the feature information collected before t_c. Moreover, the performance curve in terms of time should be smooth, which means more recent information is placed with more emphasis for predicting the performance in time slice t_c. Therefore, we estimate the feature vectors in time slice t_c by computing the weighted average of feature vectors in the past time slice:

$$\hat{p}_u(t_c) = \frac{\sum_{k=1}^{w} p_u(t_c - k) f(k)}{\sum_{k=1}^{w} f(k)}, \qquad (6)$$

$$\hat{q}_i(t_c) = \frac{\sum_{k=1}^{w} q_i(t_c - k) f(k)}{\sum_{k=1}^{w} f(k)}, \qquad (7)$$

where $\hat{p}_u(t_c)$ and $\hat{q}_i(t_c)$ are the predicted user feature vector and component feature vector in time slice t_c, respectively. w controls the information of how many past time slices are used for making prediction. In Eq. (6) and Eq. (7), large weight values are assigned to the feature vectors in recent slices while small weight values are assigned to the feature vectors in old slices.

Given the predicted latent feature vectors $\hat{p}_u(t_c)$ and $\hat{q}_i(t_c)$, we can calculate the component performance value observed by a user in time slice t_c. For the user u and the component i, the predicted performance value $\hat{r}_{ui}(t_c)$ is defined as

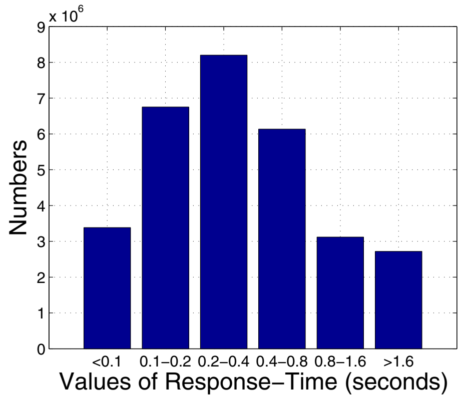
$$\hat{r}_{ui}(t_c) = \hat{p}_u^T(t_c) \hat{q}_i(t_c). \qquad (8)$$

IV. Experiments

In the following, Section IV-A gives the description of our experimental dataset, Section IV-B defines the evaluation metrics, Section IV-C evaluates the prediction quality of our approach, and Section IV-D studies the impact of data density.

Table I
Statistics of Web Service Response-Time Dataset

Statistics	Response-Time
Scale	0-20s
Mean	3.165s
Num. of Users	142
Num. of Web Services	4,532
Num. of Time Slices	64
Num. of Records	30,287,611



(a)

Figure 4. Response-Time Value Distribution

A. Dataset Description

In real world, some commercial cloud components may charge for invocations. We conduct experiments on our Web service performance dataset to evaluate the prediction quality of our proposed approach. Web service, a kind of cloud component, can be integrated into cloud applications for accessing information or computing component from a remote system. The Web service performance dataset records performance information of 4,532 real-world Web services from 57 countries. We employ 142 distributed computers located in 22 countries from PlanetLab[3] to invoke Web services. In our experiment, each of the 142 computers sends operation requests to all the 4,532 Web services in every time slice. The experiment lasts for 16 hours with one time slice lasting for 15 minutes. The response-time of all the 4,532 Web services observed by all the 142 users during 64 time slices can be presented as a set of 142×4532 matrices, each of which stands for a particular time slice.

The statistics of Web service response-time dataset are summarized in Table I. Response-time is within the range of 0-20 seconds, whose mean is 3.165 seconds. The distribution of the response-time values of all the matrices is shown in Figure 4(a). From Figure 4(a) we can observe that most of the response-time values are between 0.1-0.8 seconds.

[3]http://www.planet-lab.org

B. Metrics

We assess the prediction quality of our proposed approach in comparison with other methods by computing Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE). The metric MAE is defined as:

$$MAE = \frac{\sum_{uit} |\hat{r}_{ui}(t) - r_{ui}(t)|}{N}, \tag{9}$$

and RMSE is defined as:

$$RMSE = \sqrt{\frac{\sum_{uit} (\hat{r}_{ui}(t) - r_{ui}(t))^2}{N}}, \tag{10}$$

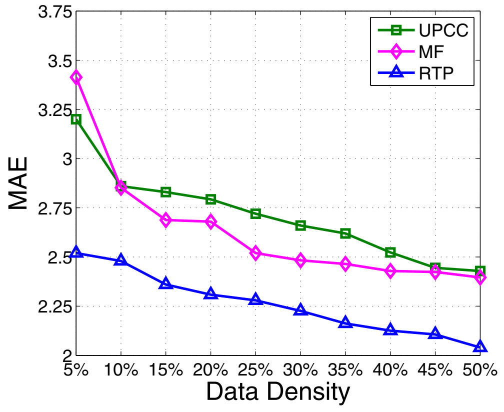
where $r_{ui}(t)$ is the response-time value of cloud component i observed by user u in time slice t, $\hat{r}_{ui}(t)$ denotes the predicted response-time value of Web service i would be observed by user u in time slice t, and N is the number of predicted response-time values in the experiments.

C. Comparison

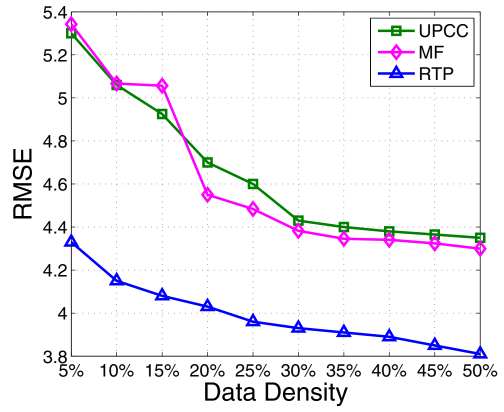
In this section, in order to show the effectiveness and efficiency of our proposed real-time cloud component performance prediction approach, we compare our approach with the following methods:

- **UPCC**-This is a neighborhood-based method. It predicts response-time of components based on the observed performance from similar users [5], [6]. Since UPCC cannot perform real-time prediction for the next time slice, we extend the traditional UPCC by using the average performance from similar users for prediction.
- **MF**-This method first compresses the set of user-component matrices into an average user-component matrix. For each entry in the matrix, the value is the average of the specific user-component pair during all the time slices. After obtaining the compressed user-component matrix, it applies the non-negative matrix factorization technique proposed by Lee and Seuing [4] on user-component matrix for missing value prediction. The predicted values are used as the response-time of the corresponding user-component pair in the next time slice.

In order to evaluate the performance of different approaches in reality, we randomly remove some entries from the performance matrices and compare the values predicted by a method with the original ones. The matrices with missing values are in different densities. For example, 10% means that we randomly remove 90% entries from the original matrices and use the remaining 10% entries for prediction. The prediction accuracy is evaluated using Eq. (9) and Eq. (10) by comparing the original values and the predicted values in the corresponding matrices. The values of λ_1 and λ_2 are tuned by performing cross-validation [7] on the observed performance data. Without lost of generality, the parameter settings of all the approaches are $l = 20$, $w = 10$, $\alpha = 1$ and $\lambda 1 = \lambda 2 = 0.001$ in the experiments



(a)



(b)

Figure 5. Performance Comparisons

conducted in this paper. Detailed impact of tensor density is studied in Section IV-D.

The component performance prediction accuracies evaluated by MAE and RMSE are shown in Figure 5. A smaller MAE or RMSE value means a better performance. From Figure 5, we can observe that our time-aware prediction method outperforms the non time-aware prediction methods (i.e., UPCC and MF), since our method employs the time-specific features as additional information for performance prediction. In Figure 5, the MAE and RMSE values of dense data (e.g., data density is 45% or 50%) are smaller than those of sparse data (e.g., data density is 5% or 10%), since denser data provide more information for prediction. On average, our real-time approach RTP improves the prediction accuracy by 18.9% and 10.3% relative to UPCC and MF, respectively. The improvements are significant, which indicates the prediction effectiveness of RTP.

D. Impact of Data Density

In Figure 5, we compare the prediction accuracy of all the methods under different data densities. We change the data density from 5% to 50% with a step value of 5%. The parameter settings in this experiment are $l = 20$, $w = 10$, $\alpha = 1$ and $\lambda 1 = \lambda 2 = 0.001$.

In Figure 5(a) and Figure 5(b), the experimental results show that our approach RTP achieves higher prediction accuracy (smaller MAE and RMSE values) than other competing methods under different data density. In general, when the data density is increased from 5% to 20%, the prediction accuracy of our approach RTP is significantly enhanced. When the data density is further increased from 20% to 50%, the enhancement of prediction accuracy will decrease. This observation indicates that collecting more performance information will greatly enhance the prediction accuracy when the data are very sparse.

V. Related Work

Cloud computing [1] has been in spotlight recently. A number of investigations have been carried out focusing on different kinds of research issues such as fault tolerance [8], resiliency quantification [9], and performance prediction [10].

Usually, performance of cloud components is measured from the user's observations. Based on the performance of components, several approaches have been proposed to optimize web component selection [11], [12], [13] in improving the whole quality of web applications.

The above approaches focus on how to employ the component performance information for component selection. However, in reality, user cannot exhaustively invoke all the components in a cloud. In this paper, we focus on predicting missing performance information by collaborative filtering approach to enable the optimal cloud component selection [14]. Typically, existed performance prediction approaches (e.g., Zhang el at. [10]) only perform static analysis on historical data while time factor is not included. To enable performance prediction in real-time Web systems, we propose a timely performance prediction framework for cloud components in this paper.

VI. Conclusion and Future Work

In this paper we propose a novel real-time cloud component performance prediction approach called RTP, for personalized performance prediction at run-time. RTP builds feature models and employs time series analysis techniques on feature trends to make personalized performance prediction for different component users. The extensive experimental results shows the effectiveness and efficiency of our framework.

For future work, we will investigate more techniques for improving the prediction accuracy (such as data smoothing and utilizing context-aware information).

Acknowledgement

The work described in this paper was substantially supported by two grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CUHK 415311 and Project No. N_CUHK405/11), and partially sponsored by the National Basic Research Program of China (973) under Grant No. 2011CB302603 and by a grant from the National Nature Science Foundation of China (No. 61100078).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] Wikipedia, "http://en.wikipedia.org/wiki/cloud_computing."

[3] G. Box, G. Jenkins, and G. Reinsel, *Time series analysis*. Holden-day San Francisco, 1970.

[4] D. Lee and H. Seung, "Learning the Parts of Objects by Non-negative Matrix Factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[5] J. Breese, D. Heckerman, C. Kadie *et al.*, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," in *Proc. of UAI'98*, 1998, pp. 43–52.

[6] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei, "Personalized QoS Prediction for Web Services via Collaborative Filtering," in *Proc. of ICWS'07*, 2007, pp. 439–446.

[7] http://en.wikipedia.org/wiki/Cross validation.

[8] Y. Zhang, Z. Zheng, and M. Lyu, "BFTCloud: A Byzantine Fault Tolerance Framework for Voluntary-Resource Cloud Computing," in *Proc. of CLOUD'11*, 2011, pp. 444–451.

[9] R. Ghosh, F. Longo, V. Naik, and K. Trivedi, "Quantifying Resiliency of IaaS Cloud," in *Proc. of SRDS'10*, 2010, pp. 343–347.

[10] Y. Zhang, Z. Zheng, and M. Lyu, "Exploring latent features for memory-based qos prediction in cloud computing," in *Proc. of SRDS'11*, 2011.

[11] T. Yu, Y. Zhang, and K. Lin, "Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints," *ACM Transactions on the Web (TWEB)*, vol. 1, no. 1, pp. 6–es, 2007.

[12] P. Bonatti and P. Festa, "On optimal service selection," in *Proc. of WWW'05*. ACM, 2005, pp. 530–538.

[13] Y. Zhang, Z. Zheng, and M. Lyu, "Wsexpress: A qos-aware search engine for web services," in *Proc. of ICWS'10*, 2010, pp. 91–98.

[14] P. Wendell, J. Jiang, M. Freedman, and J. Rexford, "Donar: decentralized server selection for cloud services," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, 2010, pp. 231–242.