An Online Performance Prediction Framework for Service-Oriented Systems

Yilei Zhang, *Student Member, IEEE,* Zibin Zheng, *Member, IEEE,* and Michael R. Lyu, *Fellow, IEEE*

***Abstract*—The exponential growth of Web service makes building high-quality service-oriented systems an urgent and crucial research problem. Performance of the service-oriented systems highly depends on the remote Web services as well as the unpredictability of the Internet. Performance prediction of service-oriented systems is critical for automatically selecting the optimal Web service composition. Since the performance of Web services is highly related to the service status and network environments which are variable over time, it is an important task to predict the performance of service-oriented systems at run-time. To address this critical challenge, this paper proposes an online performance prediction framework, called OPred, to provide personalized service-oriented system performance prediction efficiently. Based on the past usage experience from different users, OPred builds feature models and employs time series analysis techniques on feature trends to make performance prediction. The results of large-scale real-world experiments show the effectiveness and efficiency of OPred.**

***Index Terms*—Performance prediction, time series analysis, Web service.**

I. Introduction

WEB SERVICES are software systems designed to support interoperable machine-to-machine interaction over a network. With the exponential growth of Web service as a method of communications between heterogeneous systems, service-oriented architecture (SOA) is becoming a major framework for building Web systems in the era of Web 2.0 [1]. In service computing, Web services offered by different providers are discovered and integrated to implement complicated functions. Typically, a service-oriented system consists of multiple Web services interacting with each other over the Internet in an arbitrary way. How to build high-quality service-oriented systems becomes an urgent and crucial research problem.

Manuscript received February 26, 2013; accepted November 26, 2013. Date of publication January 23, 2014; date of current version August 14, 2014. This work was supported in part by the National Basic Research Program of China under 973 under Project 2014CB347701, the National Natural Science Foundation of China under Project 61100078, the Shenzhen Basic Research Program under Project JCYJ20120619153834216, and by the Research Grants Council of the Hong Kong Special Administrative Region, China, under Project CUHK 415311. This paper was recommended by Associate Editor Y. Wang. *(Corresponding author: Z. Zheng).*

The authors are with the Shenzhen Key Laboratory of Rich Media Big Data Analytics and Applications, Shenzhen Research Institute, The Chinese University of Hong Kong, and with Ministry of Education Key Laboratory of High Confidence Software Technologies (CUHK Sub-Lab), The Chinese University of Hong Kong (e-mail: zbzheng@cse.cuhk.edu.hk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2013.2297401

Good response time performance is one of the most important requirements of the service-oriented systems, which are widely employed in e-business and e-government. Typically, the response time performance of service-oriented systems involves two parts: local execution time at the system side and the response time of invoking remote Web services. While the local execution time is relatively short, the response time of invoking Web services is usually much longer, which greatly influences the system performance. The reason is that Web services are usually deployed in different geographical locations and invoked via Internet connections. Moreover, the remote Web services may be running on cheap and poor performing servers, leading to a decrease of service performance. In order to build service-oriented systems with good performance, it is important to identify Web services with low response time for composition. Moreover, by identifying the Web services with long response time at runtime, system designers can replace them with better ones to enhance the overall system performance.

Typically, Web services are considered as black boxes to service users. The user-side observed performance is employed to evaluate the qualities of Web services [2]. Since the service status (e.g., workload, CPU allocations, and so on) and the network environment (e.g., congestions, bandwidth, and so on) may change over time, response time of Web services varies a lot during different time intervals. In order to identify low response time Web services timely, real-time performance of Web services needs to be continuously monitored.

Based on the above analysis, providing real-time performance information of Web services is becoming more and more essential for service-oriented system designers to build high-quality systems and to maintain the performance of the systems at runtime. However, evaluating the performance of service-orientated systems at runtime is not an easy task, due to the following reasons.

1) Since users (SOA systems) and services are typically distributed in different geographical locations, the user-observed performance of Web services is greatly influenced by the Internet connections between users and Web services. Different users may observe quite different performance when invoking the same Web service.
2) Real-time performance evaluation may introduce extra transaction workload, which may impact the user experience of using the systems.
3) The purpose of performance evaluation is to monitor the current system performance status and allow designers

to make adjustments in order to guarantee the performance in the future. This requires frequent performance evaluation, since infrequent evaluation cannot provide useful information to designers for choosing appropriate services in the following time.

It becomes an urgent task to explore an online personalized prediction approach for efficiently estimating the performance of Web services for different service users. Based on the performance information of Web services, the overall performance of a service-oriented system can be estimated by aggregating the performance of services invoked by the system. In this paper, we propose a service performance estimation framework for providing personalized performance information to the users. The performance of services is predicted by collaborative work of users. We collect time-aware performance information from geographically distributed service users. Due to the fact that a service user usually only invokes a small number of Web services in the past and thus only observes performance of these invoked Web services, the collected performance information is usually sparse. In order to precisely predict the performance of Web service when invoked by users, we employ a set of latent features to characterize the status of Web services and users. Examples of physical feature are network distance between the user and the service server, the workload of the server, and so on. Latent features are orthogonal representation of the decomposed results of physical factors. We extract the latent features of users and services in the past time slice from the collected service performance information. By analyzing the trend of the feature changes, we estimate the features of users and services in the current time. Then the personalized performance of Web service is predicted by evaluating how the features of users apply to features of services. In summary, this paper makes the following contributions.

1) We propose an online performance prediction framework for estimating the user observed performance of service-oriented systems by employing the past usage experiences of different users to efficiently predict the performance of service-oriented systems online.
2) We collect a large-scale real-world Web service performance dataset and conduct extensive experiments for evaluating the performance of our proposed approach OPred. Totally, 4532 Web services are monitored by 142 service users and 30 287 611 invocation results are collected. Moreover, we publicly release our large-scale real-world Web service performance dataset for future research.

The rest of this paper is organized as follows. Section II describes the service-oriented system architecture and introduces the online performance prediction procedures. Sections III and IV present our online service performance prediction approach OPred in detail. Section V presents the experimental results. Section VI discusses related work and Section VII concludes the paper.

II. Preliminaries

Fig. 1 shows the architecture of a typical service-oriented system. Within a service-oriented system, several abstract tasks

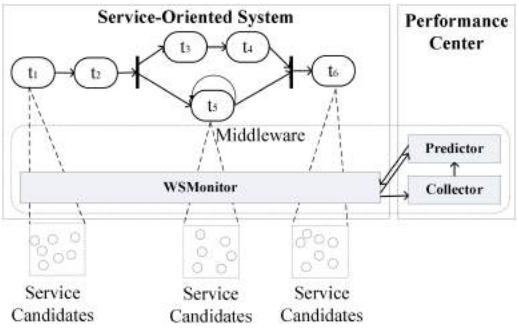


Fig. 1. Service-oriented system architecture.

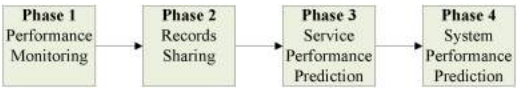


Fig. 2. Online performance prediction procedures.

are combined to implement complicated functions. For each abstract task, an optimal Web service is selected from a set of functionally equivalent service candidates. By composing the selected services, a service-oriented system instance is implemented for task execution. The problem of finding functionally equivalent Web service candidates has been discussed by a lot of previous work [3], [4], which is outside the scope of this paper. Typically the Web service candidates are provided by different organizations and 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 service-oriented systems are increasingly running on large numbers of dynamic services, users often encounter highly dynamic and uncertain performance of service-oriented systems.

As shown in Fig. 2, the online performance prediction mechanism proposed in this paper contains four phases. In phase 1, each service user keeps local performance records of the Web services. In phase 2, local Web service usage experiences are uploaded to the performance center. Each user is encouraged to contribute its local records to obtain performance prediction service from the performance center. By contributing more individually observed Web service performance records, a service user can obtain more accurate performance prediction results from the performance center. By combining performance records of several users, the performance center can obtain global performance information for all services. In phase 3, by performing time series analysis on the extracted time-specific user features and service features, a performance model is built in the performance center for personalized service performance prediction. The premise behind the performance model is that there is a small number of latent factors influencing the user observed service performance, and that a user's observed service performance is determined by how each factor applies to that user and the corresponding service at the current time slice. In phase 4, given the service level performance information, the overall performance of a service-oriented system is predicted based

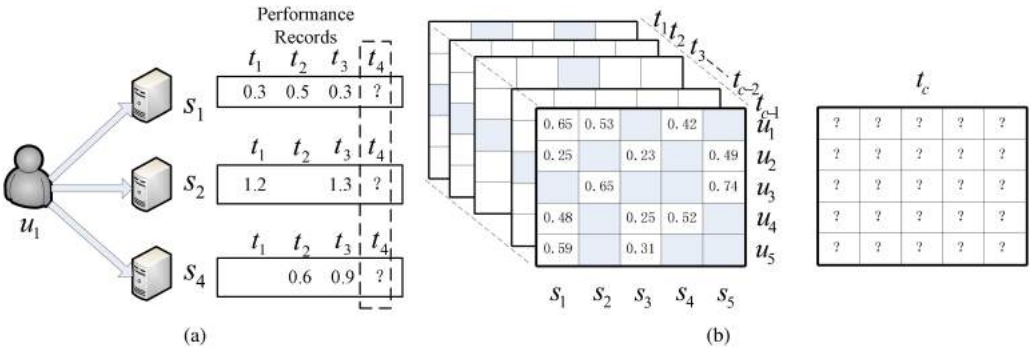


Fig. 3. Toy example of performance prediction. (a) Performance prediction. (b) Matrices of different time slices.

on the analysis of service compositional structures. When the most recent service performance information is available, an online prediction algorithm is applied for quickly updating the performance model, which requires no effort of recalculation for catching the performance trend. The detailed online service performance prediction approach is presented in Section III.

In Fig. 1, we can observe that the overall execution time of a service-oriented system mainly contains two parts: 1) local computation time at the system side and 2) response time of invoking remote services. The highly dynamic performance of service-oriented systems is mainly due to the highly dynamic response time of the composed services, while the local execution time is relatively stable. To improve the performance of systems at runtime, optimal Web service of each abstract task should be identified timely to replace the bad ones for composition. The overall performance of systems with different compositional options can be compared by estimating the total response time required for invoking all the composed services. The detailed system level performance prediction approach will be presented in Section IV.

Since most of the service users are not experts in service testing, to reduce the efforts of service users spent on testing the service performance, we design a light-weight middleware for service users to automatically record invocation results, contribute the local records to the performance center, and receive performance prediction results from the performance center. Within the middleware, there are three management components: 1) WSMonitor; 2) Collector; and 3) Predictor. WSMonitor is deployed on the user side. Collector and Predictor are deployed on the performance center. WSMonitor is responsible for monitoring the performance of Web services and sending local records to the performance center. Collector is responsible for collecting shared performance records from users. Predictor is responsible for providing time-aware personalized performance prediction based on users' performance information collected by Collector.

III. Online Service Level Performance Prediction

In this section, we propose a collaborative method to predict the performance of services. Previous Web service related techniques such as selection [5]–[8], composition [9]–[11], and orchestration [12] typically only employ average performance of service candidates at design-time. In the recent Web service literature, most of the state-of-the-art techniques can automatically update corresponding Web services with better ones at runtime. Therefore, making personalized time-specific performance prediction of Web services for different users becomes a critical task. We first formally describe the online performance prediction problem of Web services in Section III-A. Then we propose a latent feature learning algorithm to learn the time-aware user-specific and service-specific features in Section III-B. The performance of services is predicted by applying the proposed online algorithm in Section III-C. Finally, the complexity analysis is conducted in Section III-D.

A. Problem Description

Fig. 3(a) illustrates a toy example of the performance prediction problem we study in this paper. In this figure, service user u_1 has used three Web services s_1, s_2, and s_4 in the past. u_1 recorded the observed performance of Web services s_1, s_2, and s_4 with time stamp in the local site. By integrating all the performance information from different users, we can form a set of matrices as shown in Fig. 3(b) with each matrix representing a time slice. In this example, there are totally five users (from u_1 to u_5) and five services (from s_1 to s_5). Within a matrix, each entry denotes the observed performance (e.g., response time) of a Web service by a user during a specific time slice. A missing entry denotes that the corresponding user did not invoke the service in the time slice. The problem we study in this paper is how to efficiently and precisely predict performance of services observed by a user in the next time slice based on the previously collected performance information.

Let U be the set of m users and S be the set of n Web services. 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 service 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 services in time slice t_c.

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.

B. Time-Aware Latent Feature Model

In order to learn the latent features of users and services, we employ a matrix factorization technique to fit a feature model to user-service matrix in each time slice. The factorized user-specific and service-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 services by analyzing the user-service 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 service. Examples of physical features are network distance between the user and the server, the workload of the server, and so on. 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 services. Let $p(t) \in \mathbb{R}^{l\times m}$ and $q(t) \in \mathbb{R}^{l\times n}$ be the latent user and service 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 service-specific latent feature vector of a service. We employ an approximating matrix to fit the user-service 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 [13]. This matrix factorization procedure [i.e., decompose the user-service matrix $R(t)$ into two matrices $p(t)$ and $q(t)$] has clear physical meanings: Each column of $q(t)$ is a factor vector including the values of the l factors for a Web service, while each column of $p(t)$ is the user-specific coefficients for a user. In (1), the user-observed performance on service i at time t [i.e., $\hat{r}_{ui}(t)$] corresponds to the linear combination of the user-specific coefficients and the service factor vector.

In order to optimize the matrix factorization in each time slice, we first construct a cost function to evaluate the quality of approximation. The distance between two non-negative matrices is usually employed to define the cost function. 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 $R(t)$. Hence, we have the following optimization problem:

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

where $\lambda_1, \lambda_2 > 0$, I_{ui} is the indicator function that is equal to 1 if user u invoked service i during the time slice t and equal to 0 otherwise. $(r_{ui}(t) - g(\hat{r}_{ui}(t)))^2$ evaluates the error between predicted value and groundtruth value collect from real-world. To avoid the overfitting problem, we add two regularization terms to (2) to constrain the norms of $p(t)$ and $q(t)$ where $||\cdot||^2$ denotes the Frobenius norm. The optimization problem in (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]. By solving the optimization problem, we can find the most appropriate latent feature matrices $p(t)$ and $q(t)$ to characterize the users and services, respectively.

Algorithm 1: Time-Aware Latent Features Learning

Input: $R(t)$, l, λ_1, λ_2
Output: $p(t)$, $q(t)$

```
Initialize p(t) ∈ R^{l×m} and q(t) ∈ R^{l×n} with small random numbers;
Load the performance records from matrix R(t);
Calculate the objective function value L(p_u(t), q_i(t)) by Eq. (1) and Eq. (2);
repeat
    Calculate the gradient of feature vectors ∂L/∂p_u(t) and ∂L/∂q_i(t) according Eq. (3) and Eq. (4), respectively;
    Update the latent user and service feature matrices p(t) and q(t);
    p_u(t) ← p_u(t) − ∂L/∂p_u(t);
    q_i(t) ← q_i(t) − ∂L/∂q_i(t);
    Update the objective function value L(p_u(t), p_i(t)) by Eq. (1) and Eq. (2);
until Converge ;
```

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

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

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

Algorithm 1 shows the iterative process for time-aware latent feature learning. We first initialize matrices $p(t)$ and $q(t)$ with small random non-negative values. Iterations of the update rules derived from (3) and (4) allow the objective function given in (2) converge to a local minimum.

C. Service Performance Prediction

After the user-specific and service-specific latent feature spaces $p(t)$ and $q(t)$ are learned in each time slice t, we can predict the performance of a given service observed by a user during the next time slice. The service performance prediction is conducted in two phases: offline phase and online phase. In the offline phase, the performance information collected from all the service users is used for statically modeling the trends of user features and service features. By employing a time series analysis, the features of users and services in the next time slice are calculated based on the evolutionary algorithm. The predicted features are further applied for calculating the predicted performance of services in the next time slice. In the online phase, the newly observed service performance information by users at runtime is integrated into the feature model

builded in the offline phase. By employing the incremental calculation algorithm, the feature model is updated efficiently to catch the latest trend for ensuring the prediction accuracy.

Phase 1: Offline Evolutionary Algorithm: Given the latent feature vectors of users and services 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 service'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 [14]:

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

where k is the amount of time that has passed since the corresponding information was collected. $f(k)$ measures the relevance of information collect 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 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: $f(k+1) = e^{-\alpha} f(k)$, in which $e^{-\alpha}$ denotes the constant decay rate.

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 service performance observed by a user may vary significantly due to the highly dynamic service side status (e.g., workloads of weather forecasting service may increase sharply when weekends are coming) and user side environment (e.g., network latency would increase during the office hours) and 2) within a relatively short time period such as one minute or one hour, a service 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

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

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

where $\hat{p}_u(t_c)$ and $\hat{q}_i(t_c)$ are the predicted user feature vector and service feature vector in time slice t_c, respectively. w controls the information of how many past time slices are used for making prediction. In (6) and (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 predict the service performance value observed by a user in time slice t_c. For the user u and the service i, the predicted performance value $\hat{r}_{ui}(t_c)$ is defined as

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

Phase 2: Online Incremental Algorithm: In this phase, we propose an incremental algorithm for efficiently updating the feature model built in phase 1 at runtime as new performance data are collected in each time slice. In time slice t_{c-1}, $\hat{p}_u(t_{c-1})$ and $\hat{q}_i(t_{c-1})$ are predicted based on the data collected during the time slice t_{c-2-w} and t_{c-2}. During the time slice t_{c-1}, there would be some services invoked by several different users. Therefore, newly observed service performance values are available and collected from users. The new performance data are stored in a user-service matrix $R(t_{c-1})$ representing information in time slice t_{c-1}. By performing matrix factorization on $R(t_{c-1})$, latent feature vectors $p_u(t_{c-1})$ and $q_i(t_{c-1})$ in time slice t_{c-1} are learned from the real performance data. According to (6) and (7), the feature vector prediction needs to be recomputed repeatedly at each time slice using all the vectors in previous w time slices, which is highly computationally expensive. In order to predict the feature vectors in time slice t_c more efficiently, we rewrite (6) and (7) as follows:

$$\hat{p}_u(t_c) = e^{-\alpha}\left(\frac{p_u(t_{c-1})}{\sum_{k=1}^{w} f(k)} + \hat{p}_u(t_{c-1}) - \frac{p_u(t_{c-1-w}) f(w)}{\sum_{k=1}^{w} f(k)}\right) \tag{9}$$

$$\hat{q}_i(t_c) = e^{-\alpha}\left(\frac{q_i(t_{c-1})}{\sum_{k=1}^{w} f(k)} + \hat{q}_i(t_{c-1}) - \frac{q_i(t_{c-1-w}) f(w)}{\sum_{k=1}^{w} f(k)}\right) \tag{10}$$

where $e^{-\alpha}$, $f(w)$, and $\sum_{k=1}^{w} f(k)$ are constant values. $p_u(t_{c-1-w})$ and $q_i(t_{c-1-w})$ are feature vectors calculated in time slice t_{c-1-w} and can be stored with only constant memory space. $p_u(t_{c-1})$ and $q_i(t_{c-1})$ can be quickly calculated in time slice t_{c-1} since the computation complexity of matrix factorization is very low. Note that in (9) and (10), we obtain a recursive relation between $[p_u(t_{c-1}), q_i(t_{c-1})]$ and $[p_u(t_c), q_i(t_c)]$, which means the feature model in time slice t_{c-1} can be efficiently updated for predicting the feature vectors in new time slice t_c.

In the online phase, it could be possible that a new user or service is found. Since there is no prior information about the user or the service in the previous time slices, it is difficult to precisely predict the corresponding features by employing the online Incremental Algorithm. To address the cold start problem, we employ average performance for prediction. More

precisely, the prediction for a new user or a new service is set as follows:

$$\hat{r}_{ui}(t) = \begin{cases} \bar{r}_i(t), & \text{if new user and old service} \\ \bar{r}_u(t), & \text{if old user and new service} \\ \bar{r}(t), & \text{if new user and new service} \end{cases}$$

where $\bar{r}_i(t)$ is the average predicted performance of service i observed by all users in time slice t, $\bar{r}_u(t)$ is the average predicted performance of all services observed by user u in time slice t, $\bar{r}(t)$ is the average predicted performance of all user-service pairs in time slice t.

D. Computation Complexity Analysis

The offline phase includes learning latent features in w time slices and running an evolutionary algorithm. The main computation is evaluating the objective function $\mathcal{L}$ and its gradients against the variables. Since the matrix $R(t)$ is typically sparse, the computational complexity for evaluating the objective function $\mathcal{L}$ in each time slice is $O(\rho_r l)$, where ρ_r is the number of nonzero entries in the matrix $R(t)$, l is the dimension of the latent features. The computational complexities for the gradients $\frac{\partial \mathcal{L}}{\partial p_u(t)}$ and $\frac{\partial \mathcal{L}}{\partial q_i(t)}$ in (3) and (4) are $O(\rho_r l)$. Therefore, the total computational complexity in one iteration is $O(\rho_r l w)$, where w is the number of time slices. In the online phase, the main computation is factorizing the new performance matrix in time slice t. The computational complexity of online incremental algorithm is $O(\rho_r l)$.

The analysis indicates that theoretically, the computational time of offline algorithm is linear with respect to the number of observed performance entries in one time slice and the total number of time slices whose information is used for prediction. Note that because of the sparsity of $R(t)$, $\rho_r << mn$, which indicates that the computation time grows slowly with respect to the size of matrix $R(t)$. The computational time of the online algorithm is linear with the amount of newly observed performance information, which indicates that our proposed approach can efficiently integrate the performance model with new information and make online prediction timely. This complexity analysis shows that our proposed approach is very efficient and can be applied to large-scale systems.

IV. System Level Performance Prediction

In this section, we first present the aggregated response time calculation methods for basic compositional structures. Then, by analyzing the service flow, the system level response time can be predicted in a hierarchical way. The overall performance of a system consists of service response time and local execution time. Local execution time refers to the computation time between service invocations in local system. Since the variance of system performance at runtime is mainly due to the highly varying service response time, local execution time, which is relatively constant at runtime, is not included in the defined system level performance. Typically, as shown in Fig. 4, there are four types of basic compositional structures, i.e., sequence, branch, loop, and parallel. The response time of each structure can be calculated by aggregating the response time of its sub-tasks as shown in Table I.

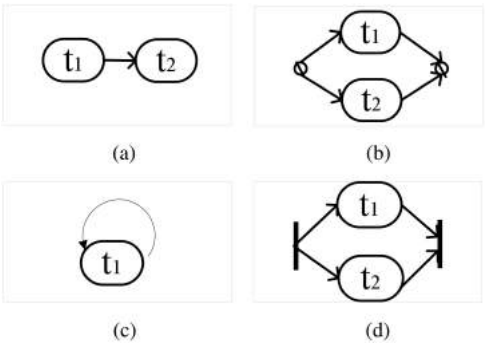


Fig. 4. Basic compositional structures. (a) Sequence. (b) Branch. (c) Loop. (d) Parallel.

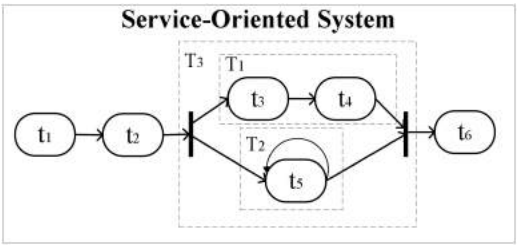


Fig. 5. Performance composition example.

TABLE I
Calculation of Aggregated Response Time

Structure	Calculation Method	Meaning of Notation
Sequence	$r = \sum_{i=1}^{n} r_i$	n: number of sequential sub-tasks r_i: response time of the i^{th} sub-task
Branch	$r = \sum_{i=1}^{n} p_i r_i$	n: number of branches r_i: response time of the i^{th} branch p_i: probability of the i^{th} branch to be executed
Loop	$r = \sum_{i=1}^{n} p_i r_i i$	n: maximum looping times r_i: response time of the i^{th} sub-task p_i: probability of executing the sub-task for i times
Parallel	$r = \max_{i=1}^{n} r_i$	n: number of branches r_i: response time of the i^{th} branch

For predicting the overall execution time of a service flow, we first decompose the system structure to a set of basic compositional structures in a hierarchical way. Then the end-to-end system execution time is calculated in a bottom up way. Take Fig. 5 as an example. First the execution time of basic compositional structures T_1 and T_2 is calculated by employing the aggregation methods of sequence and loop, respectively. Then the execution time of T_3 is calculated by employing aggregation method for branch compositional structure. Finally, the overall system execution time is calculated by employing aggregation method for sequence on t_1, t_2, T_3, and t_6.

With the aggregation approach discussed above, designers of service-oriented systems can estimate the performance of systems at design-time. At runtime, the user observed system level performance can be efficiently predicted automatically.

Once the system performance is decreased at runtime, by analyzing the system structure in a top down way, bad performance services can be quickly identified. With the predicted service performance information, dynamical service composition techniques can be employed to improve the system performance by replacing the long response time services with better ones.

V. Experiment

In this section, we conduct two experiments to evaluate our online performance prediction approach. First, by comparing with several state-of-the-art service performance prediction methods, we present the effectiveness and efficiency of our approach. Secondly, we study the service flow of a real-world service-oriented system. We also study the performance improvement by integrating the predicted performance information of our approach into the dynamic composition mechanism.

A. Service Level Evaluation

In the following, Section V-A1 introduces the experimental setup and gives the description of the experimental dataset. Section V-A2 defines the evaluation metrics. Section V-A3 compares the prediction quality of our approach with other competing approaches. Sections V-A4, V-A5, and V-A6 study the impact of data density, dimensionality, and parameter α and w, respectively. Section V-A7 compares the computational time of different approaches.

1) *Experimental Setup and Dataset Collection:* To evaluate the service level performance prediction quality of our proposed approach in the real world, we implement a tool WSMonitor for collecting the performance information of Web services. WSMonitor is deployed as a middleware on the user-side, which can continuously monitor the user experienced performance of invoked services. By sharing the user side observed performance to performance center, it can obtain performance prediction service from performance center at runtime.

WSMonitor is implemented and deployed with JDK 6.0, Eclipse 3.3, Axis 2, and Apache 2.2.17. Within WSMonitor there are several modules including WSDL Crawler, Code Generator, and Performance Monitor. WSDL Crawler first crawls a set of WSDL files from the Internet and generates a list of openly-accessible Web services. For each Web service, Code Generator automatically generates a java class for service invocation by employing the WSDL2Java tool from the Axis package [15]. Totally, 5871 classes are generated for 5871 Web services. By calling the functions generated by Code Generator, Performance Monitor is able to send operation requests to Web services and record the corresponding response time with time stamps.

We deploy the WSMonitor on 142 distributed computers located in 22 countries from PlanetLab,[1] which is a distributed test-bed consisting of hundreds of computers all over the world. Each computer acts as a service user by invoking the listed Web services from time to time. Totally, 4532 publicly available real-world Web services from 57 countries are monitored by each computer continuously. Of the initially selected Web services, 1339 are excluded in this experiment due to: 1) authentication required and 2) permanent invocation failure (e.g., the Web service is shutdown). In our experiment, each of the 142 computers sends operation requests to all the 4532 Web services in every time slice. The experiment lasts for 16 h with one time slice lasting for 15 m.

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

TABLE II
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

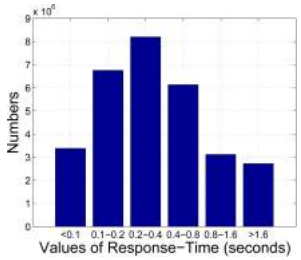


Fig. 6. Response time value distribution.

By collecting performance records from all the computers, finally 30 287 611 performance results are included into the Web service response time dataset. The response time of all the 4532 Web services observed by all the 142 service users during 64 time slices can be presented as a set of 142×4532 user-service matrices, each of which stands for a particular time slice.

The statistics of Web service response time dataset are summarized in Table II. Response-time is within the range of 0–20 s, whose mean is 3.165 s. The distribution of the response-time values of all the matrices is shown in Fig. 6. From Fig. 6 we can observe that most of the response-time values are between 0.1–0.8 s.

2) *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

$$\text{MAE} = \frac{\sum_{uit} |\hat{r}_{ui}(t) - r_{ui}(t)|}{N} \tag{11}$$

and RMSE is defined as

$$\text{RMSE} = \sqrt{\frac{\sum_{uit} (\hat{r}_{ui}(t) - r_{ui}(t))^2}{N}} \tag{12}$$

where $r_{ui}(t)$ is the response time value of Web service 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.

TABLE III
PERFORMANCE COMPARISONS (A SMALLER MAE OR RMSE VALUE MEANS A BETTER PERFORMANCE)

Data Density	RMSE	Response Time (seconds)					
		UPCC	IPCC	MF	TF	WSPred	OPred
5%	Mean	5.312	5.289	5.329	4.751	4.362	**4.330**
	Best	5.263	5.276	5.321	4.747	4.358	**4.327**
10%	Mean	5.043	4.972	5.079	4.567	4.287	**4.151**
	Best	4.962	4.946	5.063	4.563	4.283	**4.148**
45%	Mean	4.425	4.371	4.337	4.208	3.923	**3.855**
	Best	4.388	4.342	4.318	4.202	3.918	**3.851**
50%	Mean	4.352	4.354	4.298	4.016	3.899	**3.809**
	Best	4.331	4.336	4.274	4.012	3.894	**3.808**

Data Density	MAE	Response Time (seconds)					
		UPCC	IPCC	MF	TF	WSPred	OPred
5%	Mean	3.720	3.213	3.387	2.915	2.559	**2.417**
	Best	3.687	3.207	3.381	2.911	2.555	**2.413**
10%	Mean	3.264	2.841	2.873	2.786	2.495	**2.376**
	Best	3.243	2.812	2.851	2.782	2.488	**2.374**
45%	Mean	2.627	2.455	2.436	2.253	2.141	**2.029**
	Best	2.613	2.431	2.423	2.247	2.137	**2.026**
50%	Mean	2.619	2.417	2.391	2.211	2.130	**2.011**
	Best	2.609	2.404	2.384	2.207	2.125	**2.008**

3) *Comparison:* In this section, in order to show the effectiveness and efficiency of our proposed online Web service performance prediction approach, we compare the service level prediction accuracy of the following methods.

1) UPCC: This is a neighborhood-based method which employs Pearson correlation coefficient to calculate similarities between users. It predicts response time of services based on the observed performance from similar users [16], [17]. Since UPCC cannot perform online prediction for the next time slice, we extend the traditional UPCC by using the average performance from similar users for prediction.
2) IPCC: This is a neighborhood-based method which employs Pearson correlation coefficient to calculate similarities between services. It predicts response time of services based on the performance of similar services [18]. Similar to UPCC, we make an extension to IPCC in order to compare the online prediction quality with other methods.
3) MF: This method first compresses the set of user-service matrices into an average user-service matrix. For each entry in the matrix, the value is the average of the specific user-service pair during all the time slices. After obtaining the compressed user-service matrix, it applies the non-negative matrix factorization technique proposed by Lee and Seung [13] on user-service matrix for missing value prediction. The predicted values are used as the response time of the corresponding user-service pair in the next time slice.
4) TF: This is a tensor factorization based prediction method. It combines the set of user-service matrices as a tensor with a third dimension representing the time. Then it applies tensor factorization on the user-service-time tensor to extract user-specific, service-specific and time-specific characteristics. The missing value is then predicted based on how these characteristics apply to each other.
5) WSPred: This is a tensor factorization-based prediction method [19]. Different from method TF, it adds average performance value constraints when extracting the latent characteristics.
6) OPred: This method is proposed in this paper. First, the user features and service features are extracted in each time slice by employing matrix factorization. Then, the user features and service features in the new time slice are predicted by performing time analysis on the feature trends. Finally, the response time of user-service pairs is predicted by evaluating how the predicted features of users and services are applied to each other.

In order to evaluate the performance of different approaches in reality, we randomly remove some entries from the performance matrices to obtain observation matrices and compare the values predicted by a method with the original ones. The observation 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. Note that under a certain density, we employ different approaches to predict the values by using the same observation matrix. The prediction accuracy is evaluated using (11) and (12) 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 [20] on the observed performance data. Without lost of generality, the parameter settings of all the approaches are $l = 20$, $w = 8$, $\alpha = 0.2$, and $\lambda 1 = \lambda 2 = 0.001$ in the experiments conducted in this paper. Detailed impacts of parameters are studied in Sections V-A4, V-A5, and V-A6, respectively.

The service performance prediction accuracies evaluated by MAE and RMSE are shown in Table III. A smaller MAE or RMSE value means a better performance. From Table III, we

TABLE IV
PERFORMANCE IMPROVEMENT OF OPRED

Competing Approach	Performance Improvement of OPred
UPCC	22-36%
IPCC	16-25%
MF	15-28%
TF	9-17%
WSPred	1-6%

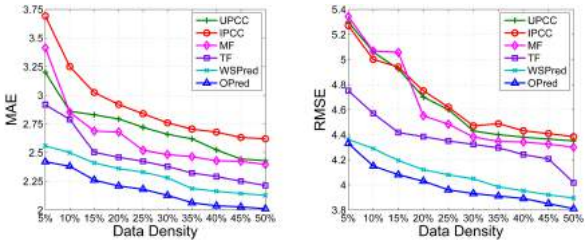


Fig. 7. Impact of data density. (a) MAE performance. (b) RMSE performance.

can observe that the time-aware prediction methods (i.e., TF and OPred) outperform the non time-aware prediction methods (i.e., UPCC, IPCC, and MF), since the time-aware methods employ the time-specific features as additional information for performance prediction. We also observe that our approach OPred constantly performs better than TF under both dense data and sparse data. This is because OPred assigns different weights on the performance information collected in different time slices. The prediction results rely more on recent user and service features than older ones. By setting $f(x)$ in (5) to a constant value (e.g., $f(x) = 1$), OPred is reduced to TF. WSPred further improves TF by employing a regularization term to prevent the predicted values from varying a lot against the average performance value. WSPred catches the periodic features of service performance. OPred proposed in this paper captures not only the periodic features but also the non-periodic features of service performance. Therefore, OPred can predict the performance trend more precisely than WSPred. Moreover, WSPred is not an online approach and requires more computational time than OPred. The computational time is compared in Section V-A7. In Table III, 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. Performance improvement of OPred is shown in Table IV. Our online approach OPred improves the prediction accuracy by 22–36%, 16–25%, 15–28%, 9–17%, and 1–6% relative to UPCC, IPCC, MF, TF, and WSPred, respectively. The improvements are significant, which indicates the prediction effectiveness of OPred.

4) *Impact of Data Density:* In Fig. 7, 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 $l = 20$, $w = 8$, $\alpha = 0.2$ and $\lambda 1 = \lambda 2 = 0.001$.

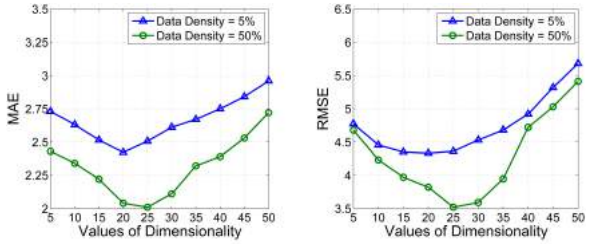


Fig. 8. Impact of dimensionality. (a) MAE performance. (b) RMSE performance.

In Fig. 7(a) and (b), the experimental results show that our approach OPred 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 OPred 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 when the data are very sparse, collecting more performance information will greatly enhance the prediction accuracy.

5) *Impact of Dimensionality:* The parameter dimensionality l determines the number of latent features applied to characterize users and services. In Fig. 8, we study the impact of parameter dimensionality by varying the values of l from 5 to 50 with a step value of five. Other parameter settings are $w = 8$, $\alpha = 0.2$, and $\lambda 1 = \lambda 2 = 0.001$.

In Fig. 8, we observe that as l increases, the MAE and RMSE decrease (prediction accuracy increases), but when l surpasses a certain threshold like 20, the MAE and RMSE increase (prediction accuracy decreases) with further increase of the value of l. This observation indicates that too few latent factors are not enough to characterize the features of user and service, while too many latent factors will cause an overfitting problem. There exists an optimal value of l for characterizing the latent features. When the data density is 50%, we observe that our approach OPred achieves the best performance when the value of dimensionality is 25, while smaller values like 5 or larger values like 50 can potentially reduce the prediction accuracy. When the data density is 5%, we observe that the prediction accuracy of our approach OPred achieves the best performance when the value of dimensionality is 20, while smaller values like 5 or larger values like 50 can potentially reduce the prediction accuracy. This observation indicates that when the service performance data are sparse, 20 latent factors are already good enough to characterize the features of user and service, which are mined from the limited performance information. On the other hand, when the data are dense, more latent factors, like 25, are needed to characterize the latent features since more performance data are available.

6) *Impact of α and w:* The parameter α controls the decaying rates of weights assigned to different time slices. A larger value of α gives more weights to the recent time slices. w controls the information of how many past time slices are used for making prediction. In Fig. 9, we vary the values of w

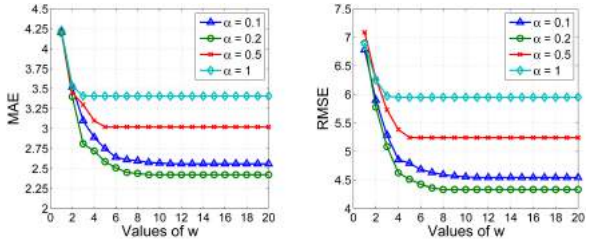


Fig. 9. Impact of α and w. (a) MAE performance. (b) RMSE performance.

from 1 to 20 with a step value of 1. Other parameter settings are $\lambda 1 = \lambda 2 = 0.001$.

Fig. 9 shows the impacts of α and w on MAE and RMSE. We observe that as w increases, the values of MAE and RMSE decrease (prediction accuracy increase) at first, but when w pass a certain threshold, the MAE and RMSE converge. This phenomenon coincides with the intuition that employing past performance information from more time slices can increase prediction accuracy. When w surpasses a certain threshold, the MAE and RMSE decrease little with further increase of the value of w. The reason is that when w is large enough, small weight values are assigned to the information of older time slices, which contribute little to the prediction accuracy. This observation indicates that too large w is unnecessary. The thresholds are different under different values of α. Since a larger value of α gives more weights to the recent time slices, the threshold is smaller than those under smaller values of α. In Fig. 9, OPred achieves the best performance when $\alpha = 0.2$. The observation confirms with the intuition that with a large value of α useful information from older time slice will be lost, and with a small value of α noisy data will cause the decrease of prediction accuracy.

7) *Computational Time Comparisons:* In Section III-D, we theoretically analyze the computation time of OPred. In this section, we compare the computation efficiencies of different approaches. In our experiments, one time slice lasts for 15 min. We compare the average computational time of a prediction approach with the length of a time slice. The data used for performance prediction are the same for all approaches. From Table V, we observe that the computational time of OPred takes less than 2% of a time slice. This observation is consistent with the time complexity analysis in Section III-D, and shows that our proposed approach OPred is efficient and can be applied to large-scale systems in real-world. TF and WSPred use more than 10% of a slice time to conduct prediction, since they are not online approaches and need to rebuild the model whenever new data are available. TF performs better than WSPred because WSPred contains an extra term in the objective function representing the average performance constraints. MF performs better than TF and WSPred because time factor is not considered when predicting the performance values. UPCC and IPCC perform worst since they are neighborhood-based approaches and take a lot of time to find the relationship between users and services.

TABLE V
AVERAGE COMPUTATIONAL TIME COMPARISONS

Approach	Computational Time	Percentage of A Time Slice
UPCC	10.095m	67.3%
IPCC	9.735m	64.9%
MF	1.575m	10.5%
TF	1.860m	12.4%
WSPred	2.055m	13.7%
OPred	0.240m	1.6%

B. System Level Performance Case Study

In this section, we evaluate our approach OPred by using a sample service-oriented system. Fig. 10 shows a typical online shopping system. It allows customers to browse and order products from the shopping website. In this shopping system, the designer integrates three Web services for providing users access to various product suppliers, banks and shippers. This example is taken from the online services provided by a gift website [21].

The service flow is illustrated in Fig. 10. By sending product queries to suppliers, the shopping system can obtain plenty of product information, which allows customers to browse various products on the website. Once a customer decides to buy a product, the shopping system sends an order request with product information to the corresponding supplier. The supplier then reserves a product for the customer and replies the shopping system with an order confirmation request. At this point, the shopping system needs to send an order confirmation to the supplier and an order request to a shipper service. Once the shopping system receives payment requests from both the product supplier and a shipper service, it proceeds to launch a payment transaction via a credit card payment service (e.g., PayPal). In the task of paying bills, customer's credit card information is transferred to the bank, and an invoice is sent back by the bank. Finally, the product supplier is notified of an bank invoice to complete the purchase. At the same time, a request is sent to the shipper to arrange the shipment of the product. Once the product is aboard, the shipper notifies the shopping system with estimated arrival date of the shipment.

We find a set of functional identical Web services from the performance dataset in Section V-A for each abstract task in the shopping system. The predicted service performance results from Section V-A are used to predicting the end-to-end performance of shopping system by employing the compositional methods in Section IV. As discussed before, by calculating system performance, poor services can be identified in a hierarchical way. Then the identified services can be replaced with better ones to maintain the overall system performance at runtime. In Fig. 11, we compare the system performance of static composition and dynamical composition. In static composition, for each abstract task we randomly choose a service from the set of functional identical candidates. The set of selected services is fixed in all time slices. In dynamical composition, the predicted service performance of OPred is employed to select the optimal services for task executions in each time slice. The comparison begins from time slice 11 since the performance information of the first 10 time slices is used as training data for OPred. The system performance

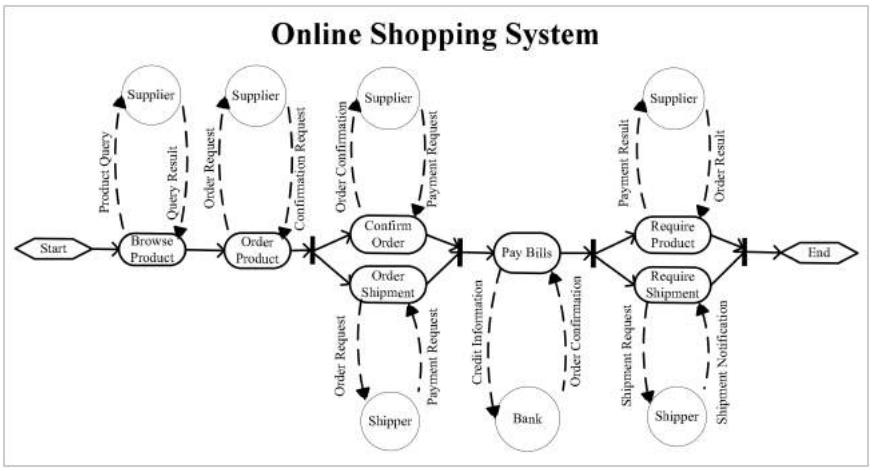


Fig. 10. Online shopping system.

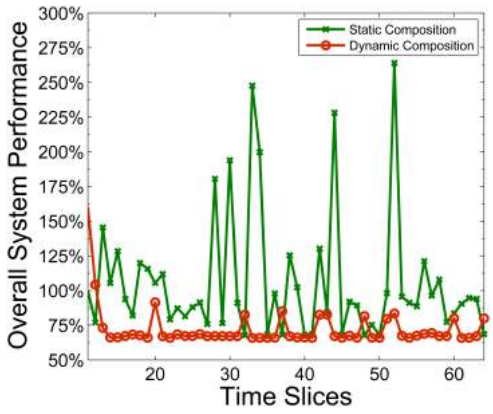


Fig. 11. System performance improvement of dynamically service composition.

of static composition method in time slice 11 is chosen as baseline. Other performance is compared with baseline in percentage (a smaller number means better performance). From Fig. 11, we can observe that the system performance of static composition is unstable at runtime. This is because the performance of some selected services is unstable, which impacts the system overall performance. For dynamic composition, since OPred can precisely predict service performance, the service-oriented system can be updated by integrating potentially optimal services at runtime. The system performance of dynamical composition maintains stable in a good level, which indicates the effectiveness of OPred.

VI. Related Work

Service-oriented systems have been in spotlight recently. SOA has become a popular framework for building service-oriented systems. A number of research investigations are focusing on different kinds of issues such as Web service selection [5]–[8], Web service composition [9]–[11], failure prediction [22], reliability prediction [23], [24], and so on. Traditionally, reliability of a system [25] is analyzed without considering the system performance, which is not accurate when applied to modern systems. Service performance can be measured either from the service provider's perspective or from the user-side. However, performance observed by different users may vary significantly due to the unpredictable communication links and heterogeneous network environments. Based on the performance of Web services, several approaches have been proposed for Web service selection [7], [8], which enable service users to identify optimal Web service from a set of functionally similar or identical Web service candidates for improving the whole quality of service-oriented systems.

The above approaches usually assume that the user-dependent performance is already known. To obtain the performance information, user-side Web service evaluations are required [26]. However, in reality a user typically has engaged a limited number of Web services in the past and cannot exhaustively invoke all the available Web service candidates. In this paper, we focus on predicting service performance in the future time slices by collaborative filtering approaches to enable the optimal Web service selection.

Collaborative filtering approaches are widely adopted in commercial recommender systems [27]. Generally, traditional recommendation approaches can be categorized into two classes: memory-based and model-based. Memory-based approaches, also known as neighborhood-based approaches, are one of the most popular prediction methods in collaborative filtering systems. Memory-based methods employ similarity computation with past usage experiences to find similar users and services for making the performance prediction. The most analyzed examples of memory-based collaborative filtering include user-based approaches [28], [29], item-based approaches [30], [31], and their fusion [23]. In [23], the reliability of an active user is predicted based on the reliability of similar users found. However, the method proposed in [23] only considers two dimensions (i.e., user and Web service) while time factor is not included. Moreover, the high computational complexity makes it difficult to extend memory-based approaches to handle large amounts of time-aware performance data for timely prediction.

Model-based approaches employ machine learning techniques to fit a predefined model based on the training datasets. Model-based approaches include several types: the clustering models [32]; the latent factor models [33]; the aspect

models [34]; and so on. Lee and Seung [13] presented an algorithm for non-negative matrix factorization that is able to learn the parts of facial images and semantic features of text. It is noted that there is only a small number of factors influencing the service performance in the user-service matrices, and that a user's factor vector is determined by how much each factor applies to that user. For a set of user-service matrices data, 3-D tensor factorization techniques are employed for item recommendation [35].

The memory-based approaches employ the information from similar users and services for predicting missing values. When the number of users or services is too small, similarity computation for finding similar users or services is not accurate. When the number of users or services is too large, calculating similarity values for each pair of users or services is time-consuming. In contrast, model-based approaches are very efficient for missing value prediction, since they assume that only a small number of factors influence the service performance. In this paper, we take advantage of a model-based method and extend it to set of user-service matrices data. The proposed method is efficient in predicting the service performance in the future time slices as analyzed in Section III-D.

VII. Conclusion and Future Work

Based on the intuition that a user's current Web service performance usage experience can be predicted by using the past usage experience from different users, we propose a novel online service performance prediction approach, called OPred, for personalized performance prediction at runtime. Using the past Web service usage experience from different users, OPred builds feature models and employs time series analysis techniques on feature trends to make personalized performance prediction for different service users. The predicted service performance is critical for identifying poor services and maintaining the system performance timely. The extensive experimental results show that our proposed OPred outperforms the state-of-the-art performance prediction approaches in terms of prediction accuracy. The case study on a typical shopping system shows the effectiveness of OPred.

For future work, we will investigate more techniques for improving the prediction accuracy (e.g., data smoothing, utilizing content-aware information, and so on). We will conduct experiments on more real-world service-oriented systems to evaluate the effectiveness and efficiency of OPred when applied to different domains.

References

[1] T. O'Reilly, "What is Web 2.0: Design patterns and business models for the next generation of software," *Commun. Strat.*, vol. 65, no. 1, p. 17, 2007.

[2] Z. Zheng, Y. Zhang, and M. R. Lyu, "Investigating QoS of real-world web services," *IEEE Trans. Serv. Comput.*, to be published.

[3] N. Salatge and J. Fabre, "Fault tolerance connectors for unreliable web services," in *Proc. IEEE/IFIP Int. Conf. DSN*, 2007, pp. 51–60.

[4] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "QoS-Aware middleware for web services composition," *IEEE Trans. Softw. Eng.*, vol. 30, no. 5, pp. 311–327, May 2004.

[5] P. Xiong, Y. Fan, and M. Zhou, "QoS-aware web service configuration," *IEEE Trans. Syst., Man Cybern. A, Syst., Humans*, vol. 38, no. 4, pp. 888–895, Jul. 2008.

[6] L. Zhang, S. Cheng, C. Chang, and Q. Zhou, "A pattern-recognition-based algorithm and case study for clustering and selecting business services," *IEEE Trans. Syst., Man Cybern. A, Syst., Humans*, vol. 42, no. 1, pp. 102–114, Jan. 2012.

[7] J. El Haddad, M. Manouvrier, and M. Rukoz, "TQoS: Transactional and QoS-aware selection algorithm for automatic web service composition," *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 73–85, Jan./Mar. 2010.

[8] T. Yu, Y. Zhang, and K. Lin, "Efficient algorithms for web services selection with end-to-end QoS constraints," *ACM Trans. Web*, vol. 1, no. 1, article 6, 2007.

[9] P. Xiong, Y. Fan, and M. Zhou, "A Petri net approach to analysis and composition of web services," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 40, no. 2, pp. 376–387, Mar. 2010.

[10] M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for QoS-based web service composition," in *Proc. Int. Conf. WWW*, 2010, pp. 11–20.

[11] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proc. Int. Conf. WWW*, 2009, pp. 881–890.

[12] D. Fensel, F. Facca, E. Simperl, and I. Toma, "Web service modeling ontology," in *Semantic Web Services*, New York, NY, USA: Springer, 2011, pp. 107–129.

[13] D. Lee and H. Seung, "Learning the parts of objects by non-negative matrix factorization," *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[14] N. N. Liu, M. Zhao, E. Xiang, and Q. Yang, "Online evolutionary collaborative filtering," in *Proc. 4th Conf. RecSys*, 2010, pp. 95–102.

[15] Apache. *Apache Axis2/Java* (2013, Jun. 1) [Online]. Available: http://axis.apache.org/axis2/java/core

[16] J. Breese, D. Heckerman and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proc. Conf. UAI*, 1998, pp. 43–52.

[17] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei, "Personalized QoS prediction for web services via collaborative filtering," in *Proc. IEEE ICWS*, 2007, pp. 439–446.

[18] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *Proc. ACM Conf. CSCW*, 1994, pp. 175–186.

[19] Y. Zhang, Z. Zheng, and M. Lyu, "Wspred: A time-aware personalized QoS prediction framework for web services," in *Proc. ISSRE*, 2011, pp. 210–219.

[20] Wikipedia. *Cross Validation* (2013, Jun. 1) [Online]. Available: http://en.wikipedia.org/wiki/Cross_validation

[21] Bjqad. *Gift Website* (2013, Jun. 1) [Online]. Available: http://bjqad.com/yawen/mall/index.asp

[22] C. Bird, N. Nagappan, H. Gall, B. Murphy, and P. Devanbu, "Putting it all together: Using socio-technical networks to predict failures," in *Proc. ISSRE*, 2009, pp. 109–119.

[23] Z. Zheng and M. Lyu, "Collaborative reliability prediction of service-oriented systems," in *Proc. ACM/IEEE ICSE*, 2010, pp. 35–44.

[24] Z. Zheng and M. R. Lyu, "Personalized reliability prediction of web services," *ACM Trans. Softw. Eng. Methodol.*, vol. 22, no. 2, p. 12, 2013.

[25] M. Lyu, *Handbook of Software Reliability Engineering*. Hightstown, NJ, USA: McGraw-Hill, 1996.

[26] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Trans. Serv. Comput.*, vol. 4, no. 2, pp. 140–152, Apr./Jun. 2011.

[27] V. Zheng, Y. Zheng, X. Xie, and Q. Yang, "Collaborative location and activity recommendations with GPS History data," in *Proc. Int. Conf. WWW*, 2010, pp. 1029–1038.

[28] Y. Shi, M. Larson, and A. Hanjalic, "Exploiting user similarity based on rated-item pools for improved user-based collaborative filtering," in *Proc. ACM Conf. RecSys*, 2009, pp. 125–132.

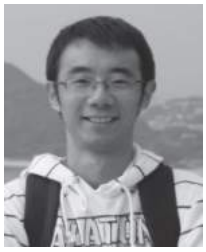
[29] W. Chen, J. Chu, J. Luan, H. Bai, Y. Wang, and E. Chang, "Collaborative filtering for Orkut communities: Discovery of user latent behavior," in *Proc. Int. Conf. WWW*, 2009, pp. 681–690.

[30] M. Deshpande and G. Karypis, "Item-based top-n recommendation algorithms," *ACM Trans. Inf. Syst.*, vol. 22, no. 1, pp. 143–177, 2004.

[31] M. Jamali and M. Ester, "TrustWalker: A random walk model for combining trust-based and item-based recommendation," in *Proc. ACM SIGKDD Conf. KDD*, 2009, pp. 397–406.

[32] G. Xue, C. Lin, Q. Yang, W. Xi, H. Zeng, Y. Yu, *et al.*, "Scalable Collaborative Filtering Using Cluster-Based Smoothing," in *Proc. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2005, pp. 114–121.

[33] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proc. Adv. NIPS*, vol. 20, 2008, pp. 1257–1264.
[34] P. Singla and M. Richardson, "Yes, there is a correlation: From social networks to personal behavior on the web," in *Proc. Int. Conf. WWW*, 2008, pp. 655–664.
[35] S. Rendle and L. Schmidt-Thieme, "Pairwise interaction tensor factorization for personalized tag recommendation," in *Proc. Int. Conf. WSDM*, 2010, pp. 81–90.



Yilei Zhang (S'10–M'13) received the B.Sc. degree in computer science from the University of Science and Technology of China, Hefei, China, in 2009. He is currently pursuing the Ph.D. degree at the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. His current research interests include service computing and cloud computing.

Mr. Zhang served as a Reviewer for a number of international journals as well as conferences including TSE, TSC, WWW, WSDM, KDD, and SCC.

Zibin Zheng (M'07) received the B.Eng. and M.Phil. degrees from Sun Yat-sen University, China, in 2005 and 2007, respectively, and the Ph.D. degree from The Chinese University of Hong Kong, Shatin, Hong Kong, in 2010.

He is currently an Associate Research Fellow at Shenzhen Research Institute, The Chinese University of Hong Kong. His current research interests include service computing and cloud computing.

Dr. Zheng is a recipient of the Outstanding Ph.D. Thesis Award of The Chinese University of Hong Kong, in 2012, the ACM SIGSOFT Distinguished Paper Award at ICSE 2010, the Best Student Paper Award at ICWS 2010, and the IBM Ph.D. Fellowship Award in 2010.

Michael R. Lyu (S'84–M'88–SM'987–F'04) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1981; the M.S. degree in computer engineering from the University of California, Santa Barbara, CA, USA, in 1985; and the Ph.D. degree in computer science from the University of California, Los Angeles, CA, USA, in 1988.

He is currently a Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, Hong Kong. His current research interests include software reliability engineering, distributed systems, and service computing.

Dr. Lyu is a fellow of AAAS for his contributions in software reliability engineering and software fault tolerance.