Online QoS Prediction for Runtime Service Adaptation via Adaptive Matrix Factorization

Jieming Zhu, Pinjia He, Zibin Zheng, *Senior Member, IEEE,* and Michael R. Lyu, *Fellow, IEEE*

Abstract—Cloud applications built on service-oriented architectures generally integrate a number of component services to fulfill certain application logic. The changing cloud environment highlights the need for these applications to keep resilient against QoS variations of their component services so that end-to-end quality-of-service (QoS) can be guaranteed. Runtime service adaptation is a key technique to achieve this goal. To support timely and accurate adaptation decisions, effective and efficient QoS prediction is needed to obtain real-time QoS information of component services. However, current research has focused mostly on QoS prediction of working services that are being used by a cloud application, but little on predicting QoS values of candidate services that are equally important in determining optimal adaptation actions. In this paper, we propose an adaptive matrix factorization (namely AMF) approach to perform online QoS prediction for candidate services. AMF is inspired from the widely-used collaborative filtering techniques in recommender systems, but significantly extends the conventional matrix factorization model with new techniques of data transformation, online learning, and adaptive weights. Comprehensive experiments, as well as a case study, have been conducted based on a real-world QoS dataset of Web services (with over 40 million QoS records). The evaluation results demonstrate AMF's superiority in achieving accuracy, efficiency, and robustness, which are essential to enable optimal runtime service adaptation.

Index Terms—Cloud computing, runtime service adaptation, QoS prediction, online learning, adaptive matrix factorization

1 Introduction

Cloud computing enables on-demand provisioning of virtual computational resources over the Internet [11]. Nowadays, cloud platforms (e.g., Amazon EC2) have become a common place to host and deliver various online applications, including search engine, e-commerce, multimedia streaming, etc. Different from hosting applications within a local enterprise network, applications deployed in a cloud might scale across multiple geographically distributed data centers [54] to serve their users globally. However, such cloud applications encounter more difficulty in guaranteeing quality of service (QoS) due to the constantly changing cloud environment. Consequently, it is a significant task for application designers to engineer their applications with self-adaptation capabilities.

Service-oriented architecture (SOA) [37] is a modern architectural paradigm that composes component services in a loosely-coupled way to fulfill complex application logic. Each service provides a self-contained unit of functionality, running in its own process and communicating with lightweight mechanisms, often an HTTP resource API. In this paper, we broadly refer to services as both component services hosted in a cloud environment and other third-party Web services [7] accessible over the Internet. Due to the loose-coupling and dynamic-binding features of SOA, many cloud applications nowadays employ SOA as a mechanism to achieve self-adaptation (e.g., [36]). For example, Amazon's e-commerce platform is built on SOA by composing hundreds of component services hosted worldwide to deliver functionalities including item recommendation, order fulfillment, and fraud detection [19]. To generate the dynamic Web content, each request typically requires a page rendering component to invoke other aggregator services, which in turn query many other services (e.g., data stores) to construct a composite response. In many cases, components even need to invoke some third-party Web services (e.g., ads recommendation). To ensure fluid page delivery, runtime service adaptation [35] is desired to achieve resilience against potential QoS variations of component services caused by the changing cloud environment.

With the rapidly growing service market, more functionally equivalent (or similar) services become available. As a simple example, both providers CDYNE.COM and WebserviceX.NET offer equivalent Web services for querying global weather information. Applications built on SOA are able to switch component services without going offline, making it possible to exploit such equivalent services for runtime service adaptation. That is, current working services can be replaced with the corresponding candidate services at runtime to tolerate unacceptable QoS degradation (e.g., unacceptable response time) or failures. Note that we refer to working services as those being used by a cloud application, and candidate services as those having equivalent functionality with working services.

Effective runtime service adaptation requires knowledge about real-time QoS changes of each working service and its candidate services in order to make timely and accurate decisions, including: 1) when to trigger an adaptation action, 2) which working services to replace, and 3) which candidate services to employ [21]. Different from traditional component-based systems, many QoS attributes of component services, such as response time and throughput,

- *J. Zhu, P. He, and M.R. Lyu are with Shenzhen Research Institute and Department of Computer Science & Engineering, The Chinese University of Hong Kong. Email: {jmzhu, pjhe, lyu}@cse.cuhk.edu.hk.*
- *Z. Zheng is with School of Data and Computer Science, and Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University, Guangzhou, China. Email: zhzibin@mail.sysu.edu.cn*

Manuscript received 26 Jul. 2016; revised 26 Jan. 2017

need to be measured from user side (i.e., applications that invoke services), rather than from service side (i.e., service providers). For a running cloud application, working services are typically continuously invoked, whereby the corresponding QoS values can be easily acquired and thus recorded. Existing studies (*e.g.*, [10], [28], [45]) on service adaptation have proposed some effective methods to predict (e.g., by using time series models) real-time QoS values of working services, which can help determine when to trigger an adaptation action and which working services to replace.

However, to the best of our knowledge, there is still a lack of research efforts explicitly targeting on addressing how to obtain real-time QoS information of candidate services, thus making it difficult in determining which candidate services to employ in an ongoing adaptation action. A straightforward solution is to measure QoS values of candidate services in an active way (a.k.a., online testing [33]), where users periodically send out service invocations and record the QoS values experienced in reply. Unfortunately, this simple solution is infeasible in many cases, since heavy service invocations are too costly for both users and service providers. On one hand, there may exist a large number of candidate services to measure. On the other hand, QoS values frequently change from time to time, thus continuous invocations are required to acquire real-time QoS information. This would incur prohibitive runtime overhead as well as the additional cost that may be charged for these service invocations.

In this paper, the problem of QoS prediction has been formulated to leverage historical QoS data observed from different users to accurately estimate QoS values of candidate services, while eliminating the need for additional service invocations from the intended users. Inspired from collaborative filtering techniques used in recommender systems [22], we propose a collaborative QoS prediction approach, namely adaptive matrix factorization (AMF). To adapt to QoS fluctuations over time, AMF significantly extends the conventional matrix factorization (MF) model by employing new techniques of data transformation, online learning, and adaptive weights. Comprehensive experiments, as well as a case study, have been conducted based on a real-world QoS dataset of Web services (with over 40 million QoS records). Compared to the existing approaches, AMF not only makes 17%∼93% improvements in accuracy but also guarantees high efficiency and robustness, which are essential in leading to optimal runtime service adaptation. The key contributions are summarized as follows:

- This is the first work targeting on predicting QoS of candidate services for runtime service adaptation, with unique requirements on accuracy, efficiency, and robustness.
- An online QoS prediction approach, adaptive matrix factorization (AMF), has been presented and further evaluated on a real-world large-scale QoS dataset of Web services.
- Both the source code and QoS dataset have been publicly released[1] to make our results easily reproducible and to facilitate future research.

1. http://wsdream.github.io/AMF

Extended from its preliminary conference version [51], the paper makes several major enhancements: a framework for large-scale QoS collection, a unified QoS model leading to better accuracy, an experimental comparison with two additional QoS prediction methods, a case study demonstrating the practical use of AMF, a discussion highlighting potential limitations and future work, and code release in both Python and Matlab for reproducible research.

The remainder of this paper is organized as follows. Section 2 introduces the background. Section 3 overviews QoS-driven runtime service adaptation. Section 4 describes our AMF approach to online QoS prediction. We report the evaluation results in Section 5, provide a case study in Section 6, and make some discussions in Section 7. We review the related work in Section 8 and finally conclude the paper in Section 9.

2 Background

In this section, we present the background of our work from the following aspects: service redundancy, QoS attributes, collaborative filtering, and matrix factorization.

2.1 Service Redundancy

It is a trend that services are becoming more and more abundant over the Internet [6]. As the number of services grows, more functionally equivalent or similar services would become available for a given task. One may argue that the number of equivalent services offered by different providers might not grow indefinitely, because usually a small number of large providers and a fair number of medium-sized providers would be enough to saturate the market. However, as pointed out in [25], a provider might offer different SLAs for each of service instances (e.g., platinum, gold and silver SLAs). Each instance might run on a physical or virtual machine with different characteristics (CPU, memory, etc.). Also, these instances might be executed at completely different locations for the purpose of load balancing and fault tolerance (e.g., by using CDNs). As we want to differentiate between each instance, there are many candidate service instances to choose for each specific task. For example, we might assume 30 different providers for each task and we would easily have 1500 candidate service instances to consider, if each provider deploys 50 instances of its service on average. Such service redundancy forms the basis for runtime service adaptation.

2.2 QoS Attributes

Quality of Service (QoS) is commonly used to describe non-functional attributes of a service, with typical examples including response time, throughput, availability, reliability, etc. [46]. Ideally, QoS values of services can be directly specified in Service-Level Agreements (SLAs) by service providers. But it turns out that many QoS specifications may become inaccurate due to the following characteristics: 1) *Time-varying*: The constantly changing operational environment, such as varying network delays across the Internet [43] and dynamic service workloads, makes many QoS attributes (e.g., response time and throughput) delivered to users fluctuate widely over time. 2) *User-specific*: The

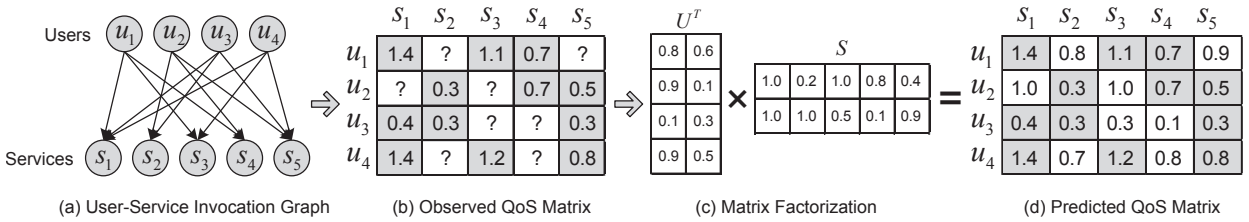


Fig. 1. An Illustration of QoS Prediction by Matrix Factorization

increase of geographic distribution of services has a non-negligible effect on user-perceived QoS. Users from different locations usually observe different QoS values even on the same service. These characteristics make obtaining accurate QoS information for runtime service adaptation become a challenging task.

2.3 Collaborative filtering

Collaborative filtering (CF) techniques [44] are widely used to rating prediction in recommender systems, such as movie recommendation in Netflix. Specifically, users likely rate the movies (e.g., 1∼5 stars) they have watched. However, for each user it is common that only a small set of movies are rated. Given a sparse user-movie rating matrix, the goal of CF is to leverage partially-observed rating data to predict remaining unknown ratings, so that proper movies can be recommended to users according to predicted ratings.

In a similar setting with rating prediction in recommender systems, historical service invocations can produce a user-service QoS matrix corresponding to each QoS attribute (*e.g.*, response time matrix). Suppose there are n users and m services, we can denote the collected QoS matrix as $R \in \mathbb{R}^{n \times m}$. Each row represents a service user (i.e. u_i), each column denotes a service (i.e. s_j) in the cloud, and each entry (i.e. R_{ij}) indicates the QoS value observed by user u_i when invoking service s_j. As shown in Fig. 1(b), the values in gray entries are QoS data observed from the user-service invocation graph in Fig. 1(a), and the blank entries are unknown QoS values to be predicted. For example, R_{11}=1.4, while R_{12} is unknown (denoted as ?). In practice, the QoS matrix is very sparse (i.e., R_{ij} is unknown in many entries), since each user often invokes only a small set of services. The QoS prediction problem can thus be modeled as a collaborative filtering problem [50], with the aim to approximately estimate the unknown values in the matrix from the set of observed entries.

2.4 Matrix factorization

Matrix factorization (MF) [40] is one of the most widely-used models to solve the above collaborative filtering problem, by constraining the rank of a QoS matrix, i.e. $rank(R) = d$. The low-rank assumption is based on the fact that the entries of R are largely correlated in practice. For instance, some close users may have similar network conditions, and thus experience similar QoS on the same service. Therefore, R has a low effective rank. Fig. 1(b)(c)(d) illustrates an example that applies matrix factorization to QoS prediction.

Concretely, the matrix R is factorized into a latent user space $U \in \mathbb{R}^{d \times n}$ and a latent service space $S \in \mathbb{R}^{d \times m}$, in such a way that R can be well fitted by the inner product of U and S, i.e. $\hat{R}_{ij} = U_i^T S_j$. To achieve this, we resolve to minimize the following loss function:

$$\mathcal{L} = \frac{1}{2} \sum_{i,j} I_{ij} (R_{ij} - U_i^T S_j)^2 + \frac{\lambda}{2} \big(\sum_i \|U_i\|_2^2 + \sum_j \|S_j\|_2^2 \big), \quad (1)$$

where the first term indicates the sum of squared error between each observed QoS value (R_{ij}) and the corresponding predicted QoS value ($U_i^T S_j$). Especially, I_{ij} = 1 if R_{ij} is observed, and 0 otherwise (e.g., I_{11} = 1 and I_{12} = 0 in Fig. 1(b)). The remaining terms, namely regularization terms [40], are added to avoid overfitting, where U_i and S_j are penalized using Euclidean norm. λ is a parameter to control the relative importance of the regularization terms.

Gradient descent [2] is the commonly adopted algorithm to derive the solutions of U and S, via an updating process starting from random initializations and iterating until convergence. After obtaining U and S, a specific unknown QoS value can thus be predicted using the corresponding inner product. For example, $\hat{R}_{12} = U_1^T S_2 = 0.8$ in Fig. 1(d).

3 QoS-Driven Runtime Service Adaptation

Fig. 2 illustrates an example of runtime service adaptation. For applications built on SOA, the application logic is typically expressed as a complex workflow (e.g., using Business Process Execution Language, or BPEL), which comprises a set of abstract tasks (e.g., A, B, C). Service composition (e.g., [15], [46]) provides a way to fulfill these abstract tasks by invoking underlying component services (e.g., A_2, B_1, C_1) provisioned in a cloud. However, in a dynamic environment, original services may become unavailable, new services may emerge, and QoS values of services may change from time to time, potentially leading to SLA (Service-Level Agreement) violations of the original service composition design. As a result, QoS-driven runtime service adaptation is desired to cope with such a changing operational environment. In the example of Fig. 2, service C_1 is replaced with service C_2 in an adaptation case that the QoS of C_1 degrades.

To support this, a framework of QoS-driven runtime service adaptation generally comprises: 1) a *BPEL engine* for workflow management and execution; 2) an *adaptation manager* that controls adaptation actions following predefined adaptation polices; 3) a *service manager* that performs service discovery and interface mediation; and 4) a *QoS manager* for QoS collection and prediction. Readers may refer to [14] for a taxonomy (plus an example framework) that provides detailed characterization of the design space of runtime adaptation meachnisms from different dimensions.

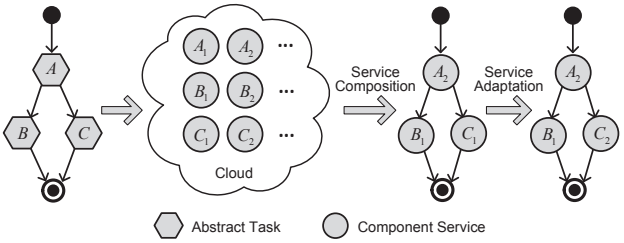


Fig. 2. An Example of Runtime Service Adaptation

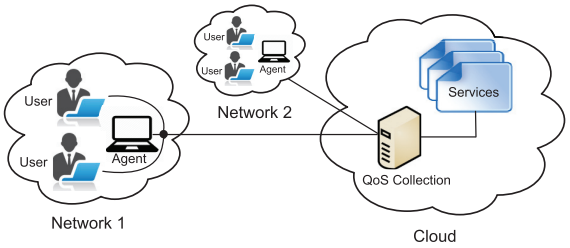


Fig. 3. A Framework for Collaborative QoS Collection

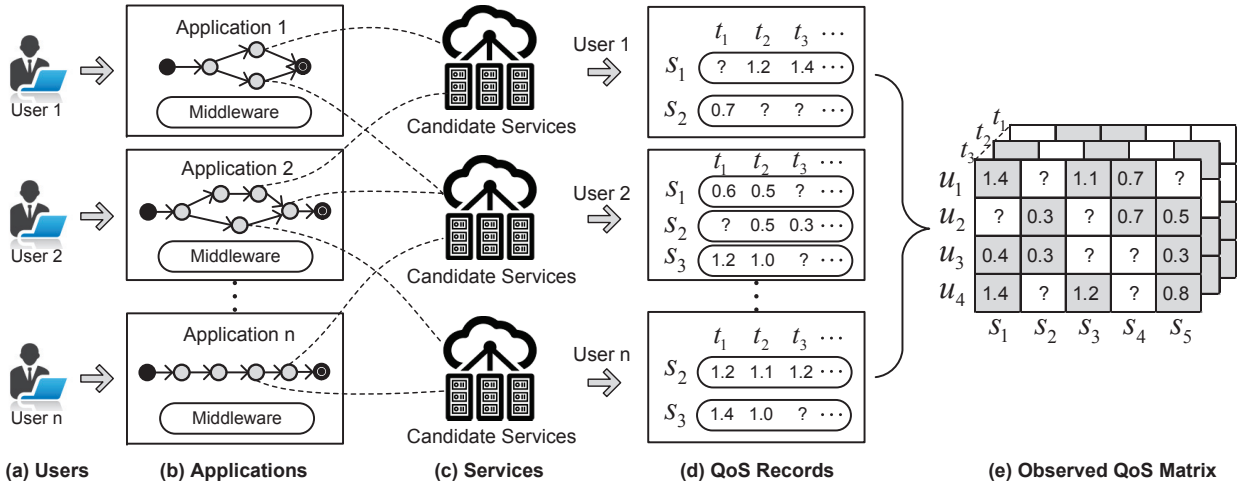


Fig. 4. The Problem Setting of Online QoS Prediction

The main focus of this paper, however, is on QoS collection and prediction.

Previously, we have developed a collaborative QoS collection framework, called WSRec [49], where users can submit historical QoS data on the services they have invoked and obtain QoS prediction results on other unused services in return. To work around the problem that some users may not want to provide their historical QoS data, we propose a new QoS collection framework as shown in Fig. 3. The framework can enrich the QoS data collection by synthetic monitoring in commercial services. For example, Dynatrace[2] provides tens of thousands of monitoring agents at a world-wide scale, across different Internet service providers for active performance measurements. We can easily take advantage of these commercial monitoring services to perform large-scale QoS measurements of services, where each synthetic user may represent a group of real users located in the same network. Within this QoS collection framework, we can make accurate QoS predictions for real users, while reducing the cost of full-mesh QoS measurements.

4 Online QoS Prediction

In this section, we formalize the online QoS prediction problem, and then highlight the challenges to address. Finally, we present our adaptive matrix factorization approach.

4.1 Problem Description

Fig. 4 illustrates the problem setting of online QoS prediction. To serve user requests, service invocations (denoted by dashed lines) are performed to fulfill certain application logic. Different applications may invoke some services in common and some other services differently according to their functionality needs. A number of candidate services exist for each component service, which allows dynamically replacing a service with another. Service invocations and adaptation actions are supported by an underlying middleware, which is able to track every service invocation and record the corresponding QoS value perceived by the user. Fig. 4(d) depicts a number of QoS records from different users during historical service invocations. Each record represents a user-observed QoS value of a service along with its timestamp. In a practical setting, we represent time as consecutive time slices. Typically, each user only invokes a small set of candidate services at one time, while many others are not invoked, leading to unknown QoS values. All the QoS records can be further assembled into a 3D (user-service-time) QoS matrix as shown in Fig. 4(e), where grey entries represent observed QoS values and blank entries are unknown. To make optimal service adaptation decisions, we need real-time QoS information of not only working services but also all candidate services. Therefore, we formulate online QoS prediction of candidate services as a problem to predict the unknown QoS values at the current time slice given all historical QoS records. By contrasting Fig. 4(e) to Fig. 1(b), we emphasize that our online QoS prediction problem differs from the existing problem setting described in Section 2.3, as we aim for an online solution to track real-time QoS values. More precisely:

2. https://www.dynatrace.com

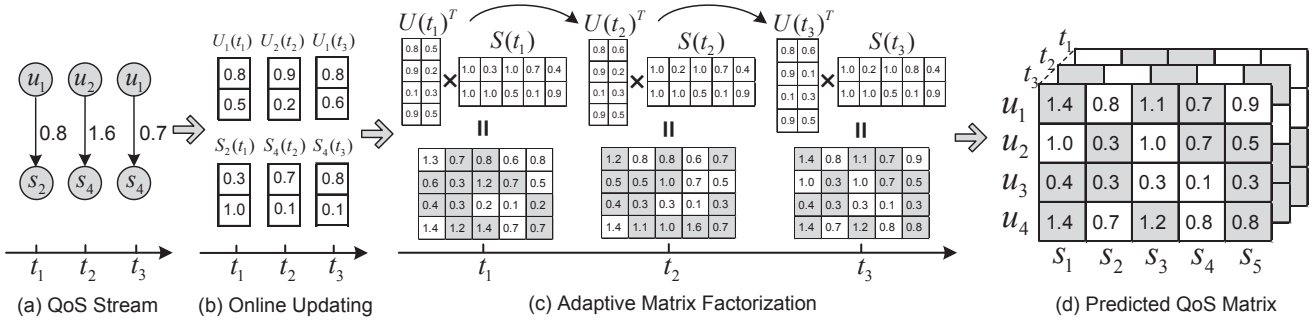


Fig. 5. An Illustration of Online QoS Prediction by Adaptive Matrix Factorization

Suppose there are n users, and m services. A QoS record $R_{ij}(t) \in \mathbb{R}^{n \times m}$ *represents the QoS value of user* u_i *invoking service* s_j *at time slice t. We denote* $I_{ij}(t) = 1$ *for an observed entry, and* $I_{ij}(t) = 0$ *for an unknown entry. Then the unknown entries* $\{R_{ij}(t_c)|I_{ij}(t_c) = 0\}$ *at current time slice* t_c *should be predicted based on all the historically observed entries* $\{R_{ij}(t)|I_{ij}(t) = 1, t \in [1, \cdots, t_c]\}$.

4.2 Challenges

Runtime service adaptation necessitates the problem of online QoS prediction with a set of unique challenges.

1) ***Accuracy***: Ensuring accuracy of QoS prediction is fundamental for runtime service adaptation. Inaccurate predictions may lead to execution of improper adaptations or missed adaptation opportunities, thus resulting in unintended SLA violations.
2) ***Efficiency***: At runtime, existing QoS values will be continuously updated with newly observed values, or become expired after a time period without updating. In order to adapt to continous QoS changes, QoS prediction should be performed efficiently in an online manner.
3) ***Robustness***: New services with different QoS values may become available, and existing services may be discontinued by their providers. Likewise, service users may often join or leave the environment. QoS prediction approaches should be robust against high churning of users and services.

These unique challenges make the online QoS prediction problem different from rating prediction in recommender systems. The conventional matrix factorization model, as described in Section 2.4, suffers from the following limitations when applied directly to online QoS prediction: *1)* While rating data typically have coherent value distributions, QoS value distributions are highly skewed with large variances, which mismatches with the underlying assumption for low-rank matrix factorization. *2)* Different from rating values that keep unchanged once being rated, QoS values are often time-varying. The conventional MF model primarily works offline and fails to adapt to continuous QoS values changes. *3)* The MF model focuses on the user-service QoS matrix with a fixed size (w.r.t. fixed users and services), which cannot handle churning of users and services without a heavy re-training process.

4.3 Adaptive Matrix Factorization

To address the above challenges, we propose a new QoS prediction approach, adaptive matrix factorization (AMF). AMF is developed based on the conventional MF model, but significantly extends it in terms of accuracy, efficiency, and robustness. Specifically, AMF integrates a set of techniques such as Box-Cox transformation, online learning, and adaptive weights. Fig. 5 presents an illustration of online QoS prediction by AMF. Different from MF shown in Fig. 1, our AMF approach collects observed QoS records as a data stream (Fig. 5(a)). The QoS stream first undergoes a data transformation process for normalization. Then the normalized QoS records are sequentially fed into the AMF model for online updating (Fig. 5(b)), which is a continuous model training process armed with online learning techniques. Intuitively, AMF works as an iterative version of MF models over consecutive time slices, as shown in Fig. 5(c), where the model trained at the previous time slice are seamlessly leveraged to bootstrap the next one. Finally, the trained model is utilized to make runtime QoS predictions (Fig. 5(d)).

4.3.1 Data Transformation

Due to our observation on a real-world QoS dataset, we find that different from the coherent value range of ratings (e.g., 1∼5 stars) in recommender systems, QoS values can vary widely (e.g., $0{\sim}20s$ for response time and $0{\sim}7000kbps$ for throughput). Moreover, the distributions of QoS data are highly skewed with large variances (see Fig. 7) compared to typical rating distributions, which mismatches with the probabilistic assumption for low-rank matrix factorization [40]. This would degrade the prediction accuracy of MF-based approaches.

To handle this problem, we apply a data transformation method, namely *Box-Cox* transformation [39], to QoS data. This technique is used to stabilize data variance and make the data more normal distribution-like in order to fit the matrix factorization assumption. The transformation is rank-preserving and performed by using a continuous function defined as follows:

$$boxcox(x) = \begin{cases} (x^{\alpha} - 1)/\alpha & \text{if } \alpha \neq 0\ , \\ \log(x) & \text{if } \alpha = 0, \end{cases} \tag{2}$$

where the parameter α controls the extent of the transformation. Note that we have $b_{max} = boxcox(R_{max})$ and $b_{min} = boxcox(R_{min})$ due to the monotonously nondecreasing property of Box-Cox transformation. R_{max} and

R_{min} are the maximal and minimal QoS values respectively, which can be specified by users (e.g., R_{max} = 20s and R_{min} = 0 for response time in our experiments). b_{max} and b_{min} are the maximal and minimal values after data transformation. Given a QoS value $R_{ij}(t)$ w.r.t. user u_i, service s_j, and time slice t, we can map it into the range $[0, 1]$ by the following normalization,

$$r_{ij}(t) = \big[boxcox(R_{ij}(t)) - b_{min}\big] / \big(b_{max} - b_{min}\big). \quad (3)$$

Especially, when $\alpha = 1$, the data transformation is relaxed to a linear normalization, where the effect of Box-Cox transformation is masked.

4.3.2 Model Formulation

In traditional QoS modeling [46], QoS attributes of services are usually defined as service-specific values, because they depend heavily on service-specific factors, such as resource capacities (e.g., CPU, memory, and I/O operations) and service loads. To capture the influence of service-specific factors, we denote q_{s_j} as the QoS bias specific to service s_j. On the other hand, many QoS attributes (e.g., response time and throughput) rely on user-specific factors, such as user locations and device capabilities. We thus denote q_{u_i} as the QoS bias specific to user u_i. In addition, QoS is highly dependent on the environment connecting users and services (e.g., the network performance). In the preliminary conference version [51], we have obtained encouraging results by modeling QoS as $U_i^T S_j$, which captures the environmental influence between user u_i and service s_j. In summary, QoS values can be jointly characterized by a user-specific part (q_{u_i}), a service-specific part (q_{s_j}), and the interaction between a user and a service ($U_i^T S_j$). Herein, we generalize the QoS model as a unified one:

$$\hat{R}_{ij}(t) = q_{u_i}(t) + q_{s_j}(t) + U_i(t)^T S_j(t), \quad (4)$$

which represents the QoS between user u_i and service s_j at time slice t. We emphasize that $\hat{R}_{ij}(t)$ is a linear combination of the above three parts without a weighting factor before each term; this is because $q_{u_i}(t)$, $q_{s_j}(t)$, and $U_i(t)^T S_j(t)$ are all unknown model parameters to be trained, which can be equally seen as the ones scaled by the corresponding weights. In particular, the above QoS model is relaxed to the one proposed in [51] when setting $q_{u_i}(t) = q_{s_j}(t) = 0$.

To fit the normalized QoS data $r_{ij}(t)$, we employ the logistic function $g(x) = 1/(1+e^{-x})$ to map the value $\hat{R}_{ij}(t)$ into the range of [0, 1], as similarily done in [40]. By applying our QoS model in Equation 4, we rewrite the loss function in Equation 1 as follows:

$$\mathcal{L}(t) = \frac{1}{2}\sum_{i,j} I_{ij}(t)\big(r_{ij}(t) - g_{ij}(t)\big)^2 \quad (5)$$
$$+\frac{\lambda}{2}\Big(\sum_i \|U_i(t)\|_2^2 + \sum_j \|S_j(t)\|_2^2 + \sum_i q_{u_i}(t)^2 + \sum_j q_{s_j}(t)^2\Big),$$

where $g_{ij}(t)$ denotes $g(\hat{R}_{ij}(t))$ for simplicity. We denote $I_{ij}(t) = 1$ if $r_{ij}(t)$ is observed, and $I_{ij}(t) = 0$ otherwise.

However, the conventional matrix factorization model, as indicated in Section 2.4, minimizes the sum of squared errors and employs the absolute error metrics, e.g., mean absolute error (MAE), for prediction accuracy evaluation. In fact, absolute error metrics are not suitable for evaluation of QoS prediction because of the large value range of QoS values. For instance, given two services with QoS values q_{s_1} = 1 and q_{s_2} = 100, the corresponding thresholds for adaptation action are set to $q_{s_1} > 5$ and $q_{s_2} < 90$. Suppose there are two sets of prediction results: (a) q_{s_1} = 8 and q_{s_2} = 99, (b) q_{s_1} = 0.9 and q_{s_2} = 92, we would choose (a) with smaller MAE. However, prediction (a) will cause a wrong adaptation action because $q_{s_1} > 5$, while prediction (b) is more reasonable. Consequently, we propose to minimize relative errors of QoS predictions, where the corresponding loss function is derived as follows:

$$\mathcal{L}(t) = \frac{1}{2}\sum_{i,j} I_{ij}(t)\Big(\frac{r_{ij}(t) - g_{ij}(t)}{r_{ij}(t)}\Big)^2 \quad (6)$$
$$+\frac{\lambda}{2}\Big(\sum_i \|U_i(t)\|_2^2 + \sum_j \|S_j(t)\|_2^2 + \sum_i q_{u_i}(t)^2 + \sum_j q_{s_j}(t)^2\Big).$$

By summing $\mathcal{L}(t)$ over all historical time slices, we intend to minimize the following global loss function: $\mathcal{L} = \sum_{t=1}^{t_c} \mathcal{L}(t)$.

4.3.3 Adaptive Online Learning

Gradient descent, as described in Section 2.4, can be used to solve the above minimization problem. But it typically works offline (with all data assembled), thus cannot easily adapt to time-varying QoS values. Online learning algorithms are required to keep continuous and incremental updating using the sequentially observed QoS data. For this purpose, we employ a widely-used online learning algorithm, stochastic gradient descent (SGD) [41], to train our AMF model. For each QoS record $(t, u_i, s_j, r_{ij}(t))$ observed at time slice t when user u_i invokes service s_j, we have the following pointwise loss function:

$$\Delta\mathcal{L} = \frac{1}{2}\Big(\frac{r_{ij} - g_{ij}}{r_{ij}}\Big)^2 + \frac{\lambda}{2}(\|U_i\|_2^2 + \|S_j\|_2^2 + q_{u_i}^2 + q_{s_j}^2), \quad (7)$$

such that $\mathcal{L} = \sum_{t=1}^{t_c}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}(t)\Delta\mathcal{L}$, the summation over all observed QoS records. Note that for simplicity, we hereafter abbreviate $r_{ij}(t), g_{ij}(t), U_i(t), S_j(t), q_{u_i}(t), q_{s_j}(t)$ by omitting the "(t)" part. Instead of directly minimizing $\mathcal{L}$, SGD relaxes to sequentially minimize the pointwise loss function $\Delta\mathcal{L}$ over the QoS stream. More precisely, with random initializations, the AMF model keeps updating on each observed QoS record $(t, u_i, s_j, r_{ij}(t))$ via the following updating rules:

$$U_i \leftarrow U_i - \eta[(g_{ij} - r_{ij})g'_{ij}S_j/r_{ij}^2 + \lambda U_i], \quad (8)$$
$$S_j \leftarrow S_j - \eta[(g_{ij} - r_{ij})g'_{ij}U_i/r_{ij}^2 + \lambda S_j], \quad (9)$$
$$q_{u_i} \leftarrow q_{u_i} - \eta[(g_{ij} - r_{ij})g'_{ij}/r_{ij}^2 + \lambda q_{u_i}], \quad (10)$$
$$q_{s_j} \leftarrow q_{s_j} - \eta[(g_{ij} - r_{ij})g'_{ij}/r_{ij}^2 + \lambda q_{s_j}], \quad (11)$$

where g'_{ij} denotes $g'(U_i^T S_j)$, and $g'(x) = e^x/(e^x+1)^2$ is the derivative of $g(x)$. η is the learning rate controlling the step size of each iteration.

As illustrated in Fig. 5(a)(b), whenever a new QoS record is observed, user u_i can take a small change on the feature vectors U_i and q_{u_i}, while service s_j can have a small change on the feature vectors S_j and q_{s_j}. The use of online learning, which obviates the requirement of retraining the whole model, enables the AMF model to adapt quickly to new QoS observations and allows for easy incorporation of new users and new services.

However, the above online learning algorithm might not perform well under the high churning rate of users and services (*i.e.*, continuously joining or leaving the environment). The convergence is controlled by the learning rate η, but a fixed η will lead to problems caused by new users and services. For example, for a new user u_1, if its feature vectors U_1 and q_{u_1} are at the initial positions, larger η is needed to help them move quickly to the correct positions. But for an existing service s_2 that user u_1 invokes, its feature vectors S_2 and q_{s_2} may have already converged. Adjusting the feature vectors of service s_2 according to user u_1, which has a large prediction error with unconverged feature vectors, is likely to increase prediction error rather than to decrease it. Consequently, our AMF model, if performed online, needs to be robust against the churning of users and services.

To achieve this goal, we propose to employ adaptive weights to control the step size when updating our AMF model. Intuitively, an accurate user should not move much according to an inaccurate service while an inaccurate user need to move a lot with respect to an accurate service, and vice versa. For example, if service s_1 has an inaccuracy of 10% and service s_2 with inaccuracy 1%, when a user invokes both s_1 and s_2, it should move less for its feature vector to service s_1 compared with service s_2. We denote the average error of user u_i as e_{u_i} and the average error of service s_j as e_{s_j}. Accordingly, we set two weights w_{u_i} and w_{s_j} for user u_i and service s_j respectively, to control the robustness between each other. Namely,

$$w_{u_i} = e_{u_i}/(e_{u_i} + e_{s_j}), \quad w_{s_j} = e_{s_j}/(e_{u_i} + e_{s_j}). \tag{12}$$

Note that $w_{u_i} + w_{s_j} = 1$. To update e_{u_i}, e_{s_j}, we apply the exponential moving average [1], which is a weighted average with more weight (controlling by β) given to the latest data.

$$e_{u_i} = \beta w_{u_i} e_{ij} + (1 - \beta w_{u_i}) e_{u_i}, \tag{13}$$
$$e_{s_j} = \beta w_{s_j} e_{ij} + (1 - \beta w_{s_j}) e_{s_j}, \tag{14}$$

where e_{ij} denotes the relative error of a QoS record, between g_{ij} and r_{ij}:

$$e_{ij} = |r_{ij} - g_{ij}|/r_{ij}. \tag{15}$$

After obtaining the updated weights w_{u_i} and w_{s_j} at each iteration, we finally refine Equation 8∼11 to the following updating rules:

$$U_i \leftarrow U_i - \eta w_{u_i}[(g_{ij} - r_{ij}) g'_{ij} S_j / r_{ij}^2 + \lambda U_i], \tag{16}$$
$$S_j \leftarrow S_j - \eta w_{s_j}[(g_{ij} - r_{ij}) g'_{ij} U_i / r_{ij}^2 + \lambda S_j], \tag{17}$$
$$q_{u_i} \leftarrow q_{u_i} - \eta w_{u_i}[(g_{ij} - r_{ij}) g'_{ij} / r_{ij}^2 + \lambda q_{u_i}], \tag{18}$$
$$q_{s_j} \leftarrow q_{s_j} - \eta w_{s_j}[(g_{ij} - r_{ij}) g'_{ij} / r_{ij}^2 + \lambda q_{s_j}]. \tag{19}$$

With U_i, S_j, q_{u_i}, and q_{s_j} obtained at current time slice t_c, we can predict, according to Equation 4, the unknown QoS value $R_{ij}(t_c)$ (where $I_{ij}(t_c) = 0$) for an ongoing service invocation between user u_i and service s_j at t_c. Meanwhile, a backward data transformation of $g(\hat{R}_{ij}(t_c))$ is required, which can be computed according to the inverse functions for Equation 2 and 3.

4.3.4 Algorithm Description

Algorithm 1 provides the pseudo code of our AMF approach. Specifically, at each iteration, the newly observed QoS data are collected to update the model (lines 2∼8), or else existing historical data are randomly selected for model updating (lines 10∼14) until convergence. Especially, according to the steps described in Section 4.3.1∼4.3.3, the set of updating operations has been defined as a function UPDATEAMF$(t, u_i, s_j, R_{ij}(t))$ with parameters set as a QoS data sample $(t, u_i, s_j, R_{ij}(t))$. For a newly observed data sample, we first check whether the corresponding user or service is new, so that we can add it to our model (lines 5∼7) and keep updating its feature vectors when more observed data are available for this user or service (line 8). As such, our model can easily scale to new users and services without re-training the whole model. Another important point is that we check whether an existing QoS record has expired (line 11), and if so, discard this value (*i.e.*, in line 14, we set $I_{ij}(t_c)$ = 0). In our experiments, for example, we set the expiration time interval to 15 minutes.

Algorithm 1 Adaptive Matrix Factorization

Input: Sequentially observed QoS stream: $\{(t, u_i, s_j, R_{ij}(t))\}$; and model parameters: λ, η, β.
Output: Online QoS prediction results: $\{\hat{R}_{ij}(t_c) | I_{ij}(t_c) = 0\}$.

▷ by Equation 4
1: **repeat** ▷ Continuous updating
2: Collect newly observed QoS data;
3: **if** *a new data sample* $(t_c, u_i, s_j, R_{ij}(t_c))$ *received* **then**
4: Set $I_{ij}(t_c) \leftarrow 1$;
▷ $R_{ij}(t_c)$ is observed at current time slice t_c
5: **if** u_i *is a new user or* s_j *is a new service* **then**
6: Randomly initialize $U_i \in \mathbb{R}^d$, or $S_j \in \mathbb{R}^d$;
7: Initialize $e_{u_i} \leftarrow 1$, or $e_{s_j} \leftarrow 1$;
8: UPDATEAMF$(t_c, u_i, s_j, R_{ij}(t_c))$;
9: **else**
10: Randomly pick a historical data sample: $(t, u_i, s_j, R_{ij}(t))$;
11: **if** $t_c - t < TimeInterval$ **then**
12: UPDATEAMF$(t, u_i, s_j, R_{ij}(t))$;
13: **else**
14: Set $I_{ij}(t_c) \leftarrow 0$;
▷ Historical data sample is obsolete
15: **if** *converged* **then**
16: Wait until new QoS data observed;
17: **until** forever

18: **function** UPDATEAMF$(t, u_i, s_j, R_{ij}(t))$
19: Normalize $r_{ij}(t) \leftarrow R_{ij}(t)$; ▷ by Equation 2 and 3
20: Update $w_{u_i} \leftarrow (e_{u_i}, e_{s_j})$, and $w_{s_j} \leftarrow (e_{u_i}, e_{s_j})$;
▷ by Equation 12
21: Compute $e_{ij} \leftarrow (r_{ij}(t), g_{ij}(t))$; ▷ by Equation 15
22: Update $e_{u_i} \leftarrow (w_{u_i}, e_{ij}, e_{u_i})$, and $e_{s_j} \leftarrow (w_{s_j}, e_{ij}, e_{s_j})$;
▷ by Equation 13 and 14
23: Update $q_{u_i}(t_c), q_{s_j}(t_c), U_i(t_c), S_j(t_c)$ simutaneously;
▷ by Equation 16∼19

5 EVALUATION

In this section, we conduct a set of experiments on a QoS dataset of real-world Web services to evaluate our AMF approach from various aspects, including prediction accuracy, efficiency, robustness, and parameter analysis. We implement our AMF algorithm as well as the baseline approaches using C++, and further wrap up the code as a Python package for ease of use. The package consists of over 1000 lines of code in total. All the experiments were conducted on a Linux server with Intel Xeon E5-2670v2

Property	Value
#QoS records	40,896,000
#Users	142
#Services	4,500
#Time slices	64
Time interval	$15min$
RT range	$0 \sim 20s$
RT average	$1.33s$
TP range	$0 \sim 7000kbps$
TP average	$11.35kbps$

Fig. 6. Data Statistics

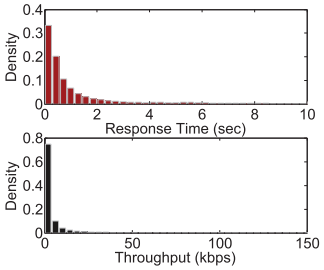


Fig. 7. Data Distribution

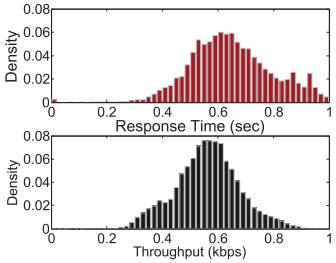


Fig. 8. Transformed Distribution

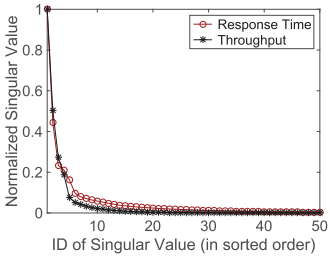


Fig. 9. Sorted Singular Values

CPU and 128G DDR3 1600 RAM, running 64-bit Ubuntu 14.04.2 with Linux kernel 3.16.0. Note that all the source code has been publicly released and well documented for reproducible research.

5.1 Data Description

In our experiments, we focus on two QoS attributes, response time (RT) and throughput (TP), both of which are important in characterizing the non-functional quality aspects of services. Response time stands for the time duration between user sending out a request and receiving a response, while throughput denotes the data transmission rate (*e.g.*, $kbps$) of a user invoking a service.

We use a publicly-available QoS dataset of real-world Web services, namely WS-DREAM [8], previously collected by our group. The dataset has been widely used in the research community since its release. It consists of about 40.9 million QoS records, with response time and throughput values recorded during the service invocations between 142 users and 4,500 Web services over 64 consecutive time slices, at an interval of 15 minutes. Specifically, the 142 users are set on PlanetLab[3] nodes distributed in 22 countries, and the services are 4,500 publicly accessible real-world Web services crawled from the Internet, which are hosted at 57 countries.

Fig. 6 provides some basic statistics of our data. Both QoS attributes have a wide value range: the response time has a range of 0∼$20s$ ($1.33s$ on average), and the throughput ranges in 0∼$7000kbps$ ($11.35kbps$ on average). We further plot the data distributions of response time and throughput in Fig. 7. For better visualization, we cut off the response time beyond $10s$ and the throughput more than $150kbps$. It is shown that the data distributions are highly skewed. In contrast, as shown in Fig. 8, we obtain more normal data distributions through our data transformation in Section 4.3.

In addition, we investigate the singular values of the data matrices of response time and throughput. The singular values are computed by singular value decomposition (SVD) [5] and then normalized so that the largest singular value is equal to 1. Fig. 9 presents the singular values in descending order. We can observe that most of the singular values, except the first few ones, are close to 0. This observation indicates that both response-time and throughput data matrices are approximately low-rank, which complies with the low-rank assumption of matrix factorization. In our experiments, we set the rank $d = 10$ by default.

3. PlanetLab (https://www.planet-lab.org) is a global open platform for distributed systems research.

5.2 Evaluation Metrics

We evaluate the prediction accuracy of our AMF approach in comparison with other baseline approaches by using the following metrics.

- **MRE** (Median Relative Error). MRE takes the median value of all the pairwise relative errors:

$$MRE = \underset{I_{ij}(t)=0}{median} \left\{ \left| \hat{R}_{ij}(t) - R_{ij}(t) \right| \Big/ R_{ij}(t) \right\}, \quad (20)$$

  where $R_{ij}(t)$ is the real QoS value and $\hat{R}_{ij}(t)$ is the corresponding predicted value.
- **NPRE** (Ninety-Percentile Relative Error). NPRE takes the 90th percentile of all the pairwise relative errors.

Due to the large variance of QoS values, our optimization efforts are focused more on relative error metrics, which indicate appropriate for QoS prediction evaluation as mentioned in Section 4.3.2.

5.3 Accuracy Comparison

In order to evaluate the effectiveness of AMF, we compare the prediction accuracy of AMF against a baseline approach and four other approaches that are deemed potential in QoS prediction [47], [48], [49], [50]. Although these approaches are not originally proposed for runtime service adaptation, we include them for comparison purposes.

- **Average**: This baseline approach uses the average QoS value at each time slice as the prediction result. The baseline is a simple QoS prediction method without any optimization incorporated.
- **UIPCC**: User-based collaborative filtering approach (namely UPCC) employs the similarity between users to predict the QoS values, while item-based collaborative filtering approach (namely IPCC) employs the similarity between services to predict the QoS values. UIPCC is a hybrid approach proposed in [49], by combing both UPCC and IPCC approaches to make full use of the similarity information between users and the similarity information between services for QoS prediction.
- **PMF**: This is a widely-used implementation of the matrix factorization model [40]. As introduced in Section 2.4, PMF has been applied to offline QoS prediction in [50]. For our online QoS prediction problem, we apply PMF to the observed QoS matrix independently at each time slice.

TABLE 1
QoS Prediction Accuracy (Smaller MRE and NPRE Indicate Better Accuracy)

QoS	Methods	MRE							NPRE						
		D=5%	D=10%	D=15%	D=20%	D=25%	D=30%	Improve	D=5%	D=10%	D=15%	D=20%	D=25%	D=30%	Improve
RT	Average	1.405	1.414	1.417	1.419	1.420	1.421	**81.2%**	7.347	7.352	7.354	7.356	7.355	7.356	**87.5%**
	UIPCC	0.748	0.644	0.580	0.551	0.533	0.518	**54.9%**	6.381	5.387	4.663	4.322	4.116	3.961	**80.5%**
	PMF	0.580	0.522	0.523	0.524	0.523	0.521	**49.9%**	2.264	2.785	2.964	3.032	3.054	3.055	**67.3%**
	WSPred	0.485	0.478	0.466	0.460	0.462	0.445	**42.8%**	3.103	2.876	2.628	2.592	2.658	2.474	**66.2%**
	NTF	0.470	0.462	0.458	0.452	0.449	0.442	**41.5%**	3.032	2.983	2.965	2.962	2.970	2.839	**69.0%**
	AMF	**0.304**	**0.275**	**0.263**	**0.257**	**0.252**	**0.249**	-	**1.014**	**0.934**	**0.908**	**0.893**	**0.883**	**0.876**	-
TP	Average	0.562	0.563	0.563	0.563	0.563	0.563	**52.8%**	7.615	7.659	7.671	7.675	7.678	7.682	**87.3%**
	UIPCC	1.500	1.434	1.373	1.262	1.161	1.097	**79.6%**	15.042	15.063	14.936	14.286	13.624	13.258	**93.2%**
	PMF	0.508	0.462	0.453	0.442	0.431	0.419	**41.5%**	1.649	2.130	2.343	2.424	2.444	2.436	**55.0%**
	WSPred	0.321	0.315	0.318	0.320	0.317	0.316	**16.5%**	2.319	2.453	2.507	2.551	2.573	2.585	**60.7%**
	NTF	0.329	0.321	0.320	0.317	0.318	0.316	**17.2%**	2.363	2.440	2.434	2.448	2.464	2.395	**59.7%**
	AMF	**0.320**	**0.278**	**0.260**	**0.250**	**0.244**	**0.241**	-	**1.135**	**1.003**	**0.956**	**0.934**	**0.920**	**0.911**	-

- **WSPred**: This approach [48] solves time-aware QoS predictions of Web services by taking into account the time information. It leverages the tensor factorization model [26], based on a generalization of low-rank matrix factorization, to represent the 3D (user-service-time) QoS matrix.
- **NTF**: Non-negative tensor factorization (NTF) is an approach recently proposed in [47], which further extends WSPred with non-negative constraints on the tensor factorization model for time-aware QoS prediction.

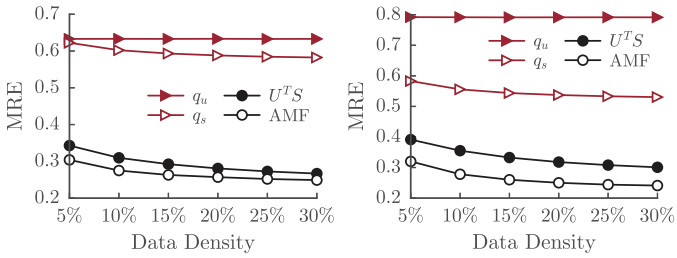
As we mentioned before, QoS matrix collected from practical usage data is usually very sparse (i.e., most of the entries $R_{ij}(t)$ are unknown), because of the large number of candidate services as well as the high cost of active service measurements. To simulate such a sparse situation in practice, we randomly remove entries of our full QoS dataset so that each user only keeps a few available historical QoS records at each time slice. In particular, we vary the data density from 5% to 30% at a step increase of 5%. Data density = 5%, for example, indicates that each user invokes 5% of the candidate services, and each service is invoked by 5% of the users.

For AMF approach, the remaining data entries are randomized as a QoS stream for training, while the removed entries are used as the testing data to evaluate the prediction accuracy. In this experiment, we set the parameters $d = 10$, $\lambda = 0.001$, $\beta = 0.3$, and $\eta = 0.8$. Especially, the parameter of Box-Cox transformation, α, can be automatically tuned using the Python API, $scipy.stats.boxcox$ [4]. Note that the parameters of the competing approaches are also optimized accordingly to achieve their optimal accuracy on our data. Under a specified data density, each approach is performed 20 times with different random seeds to avoid bias, and the average results are reported.

Table 1 provides the prediction accuracy results in terms of MRE and NPRE. We can observe that our AMF approach achieves the best accuracy results (marked in bold) among all the others. The "*Improve*" columns show the corresponding improvements averaging over different data densities, each measuring the percentage of how much AMF outperforms the other existing approach. For response time (RT) attribute, AMF achieves 41.5%∼81.2% improvements on MRE and 66.2%∼87.5% improvements on NPRE. Likewise, for throughput (TP) attribute, AMF has 16.5%∼79.6% MRE improvements and 55.0%∼93.2% NPRE improvements. In particular, the baseline method, Average, has the worst accuracy on RT data, since no optimization is incorporated. UIPCC attains better accuracy on RT, but results in the worst accuracy on TP data. This is because UIPCC, as a basic collaborative filtering approach, is sensitive to the large variance of TP data. The conventional MF model, PMF, obtains consistently better accuracy than Average and UIPCC, as reported in [50]. Meanwhile, WSPred and NTF further improve PMF by considering time-aware QoS models with tensor factorization. However, none of these existing models deals with the large variance of QoS data. They all yield high NPRE values, which is not suitable for QoS prediction problem. Our AMF model can be seen as an extension of the conventional collaborative filtering models, by taking into account the characteristics of QoS attributes, which thereby produces good accuracy results.

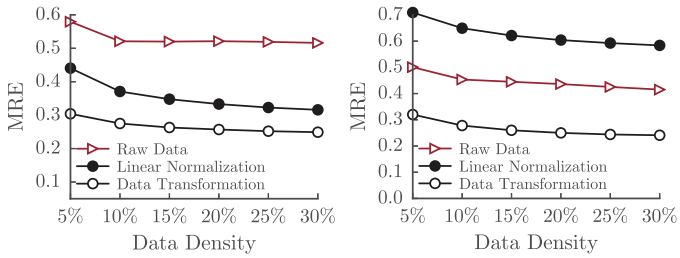
5.4 Effect of QoS Modeling

In this section, we intend to evaluate the effectiveness of the unified QoS model, as presented in Equation 4, which consists of a user-specific part (i.e., q_u), a service-specific part (i.e., q_s), and the part characterizing the interaction between a user and a service (i.e., $U^T S$). Concretely, we employ each part individually as a separate QoS model to make QoS predictions. Fig. 10 presents the MRE results corresponding to response time and throughput data. The results show that q_u produces the worst accuracy. This is because user-specific QoS modeling cannot capture the properties of QoS attributes that are more or less associated with services in practice. Meanwhile, q_s, as the conventional QoS model, considers only service-specific factors but ignores user-specific factors, thus suffering from inaccuracy in QoS modeling as well. In contrast, $U^T S$, which is the QoS model we proposed in the preliminary conference version of this work [51], obtains decent accuracy results. It is an effective QoS model because both user-specific and service-specific factors are successfully incorporated via a low-rank matrix factorization model. In this paper, based on our unified QoS model, AMF outperforms the previous version in [51], with an average of 9.2% MRE improvement on response-time data and an average of 20.7% MRE improvement on throughput data. Especially, the improvement in Fig. 10(b)



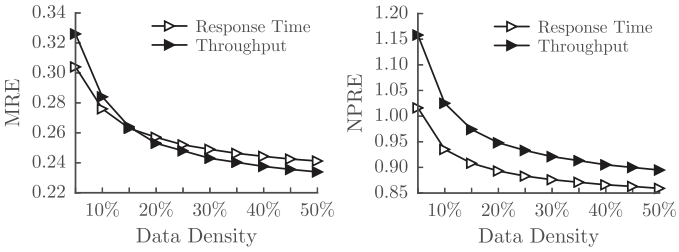
(a) Response Time (b) Throughput

Fig. 10. Effect of QoS Modeling



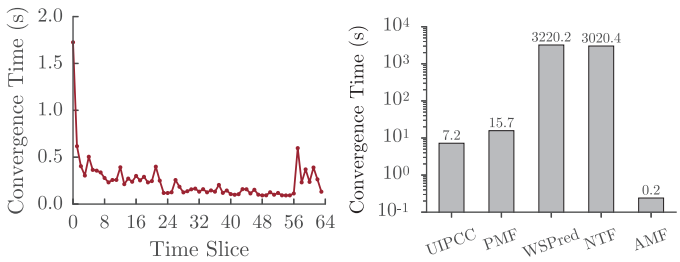
(a) Response Time (b) Throughput

Fig. 11. Impact of Data Transformation



(a) MRE (b) NPRE

Fig. 12. Effect of Data Density



(a) (b)

Fig. 13. Efficiency Evaluation Result

is larger than that in Fig. 10(a) since q_s obtains a better result on throughput data. This indicates the positive effect of q_u and q_s on accuracy improvements.

5.5 Effect of Data Transformation

In Fig. 8, as shown before, we have visualized the effect of data transformation on data distributions. To further evaluate the effect of data transformation, we compare the prediction accuracy among three different data processing methods: *raw data* without normalization, *linear normalization*, and *data transformation*. Our data transformation method works as a combination of Box-Cox transformation in Equation 2 and linear normalization in Equation 3. Especially, when $\alpha = 1$, our data transformation is relaxed to a linear normalization procedure, since the effect of the function $boxcox(x)$ is masked. In this experiment, the parameter α is automatically tuned to -0.007 for response time and -0.05 for throughput. We then vary the data density and compute the corresponding MRE values. The results are presented in Fig. 11. We can observe that the three data processing methods have different effects on prediction accuracy. More specifically, the model trained on raw data suffers from the data skewness problem, resulting in large MRE. The linear normalization method improves the prediction accuracy of response time, but has a negative effect on the prediction accuracy of throughput. We speculate that this is caused by the extreme skewness of throughput data (Fig. 7). The linear normalization narrows down the QoS value range, and to some extent exacerbates the data skewness. However, AMF mitigates this issue with Box-Cox transformation, and therefore, improves a lot in MRE.

5.6 Effect of Data Density

In our experiments, we simulate the sparse situation in reality by randomly removing QoS samples of the dataset. We set the parameter of data density (i.e., the percentage of data preserved) to control the sparsity of QoS data. To present a comprehensive evaluation of the effect of data density on prediction accuracy, we vary the data density from 5% to 50% at a step increase of 5%. We also set the other parameters as in Section 5.3. Fig. 12 illustrates the evaluation results in terms of MRE and NPRE. We can observe that the prediction accuracy improves with the increase of data density. The improvement is significant especially when the QoS data is excessively sparse (e.g., 5%). This is because the QoS prediction model likely falls into the overfitting problem given only limited training data. With more QoS data available, the overfitting problem can be alleviated, thus better prediction accuracy can be attained.

5.7 Efficiency Analysis

To evaluate the efficiency of our approach, we obtain the convergence time of AMF at every time slice while setting data density to 10%. As we can see in Fig. 13(a), after the convergence at the first time slice, our AMF approach then becomes quite fast in the subsequent time slices, because AMF keeps online updating with sequentially observed QoS records. The convergence time may sometimes fluctuate a little bit (e.g., at time slice 57), due to the change of data characteristics. We further compare the average convergence time of AMF against the other four approaches: UIPCC, PMF, WSPred, and NTF. As shown in Fig. 13(b), we can find that all these approaches are more computationally expensive than AMF, because they need to re-train the whole model at each time slice, which incurs high computational overhead compared to the online algorithm. Especially, WSPred and NTF, which are based on tensor factorization models, consume the longest time for training. Therefore, these existing approaches are more appropriate for one-time training as used in traditional recommender systems.

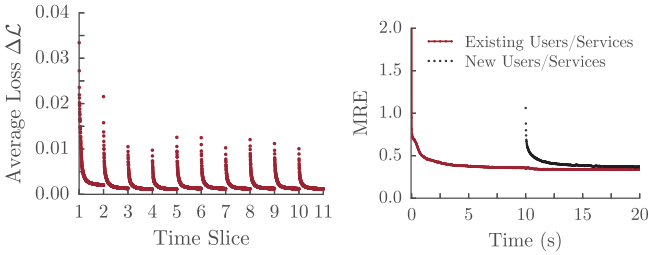


Fig. 14. Robustness Evaluation Result

Fig. 15. A Prototype Application for Flight Purchase

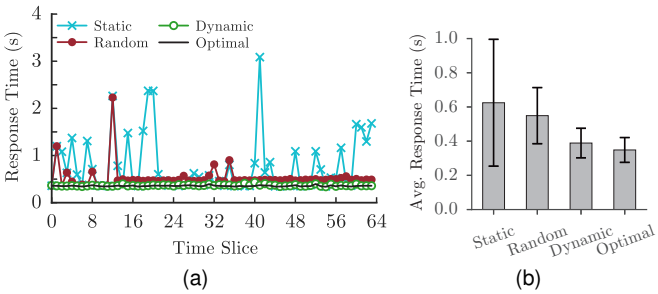


Fig. 16. Case Study Result

5.8 Robustness Analysis

In face of the churning of users and services, coupled with the rapid changes of QoS values, the AMF model needs to be efficient in adapting to new changes and robust in making accurate QoS predictions. In this section, we evaluate the prediction accuracy on these new users and services, as well as the robustness of the prediction results. For this purpose, we track the average loss value along time, while adding new QoS records of a new time slice every 10 seconds, with the data density setting to 10%. The quick convergence in Fig. 14(a) shows that AMF can adapt quickly to new QoS observations at each time slice. We then simulate the situation of new users and services using our dataset. We randomly select 80% of users and services at time slice 1 as existing users and services, and then train the AMF model using the corresponding QoS data. After the model converges, we add the remaining 20% of users and services into the model at time $t = 10$. Fig. 14(b) indicates that the MRE for the new users and services decreases rapidly after they join the model, while the MRE for existing users and services still keeps stable. This implies the robustness of our model to deal with new users and services.

6 Case Study

In this section, we evaluate the effectiveness of our online QoS prediction approach via a case study. Whereas a complex application workflow is generally composed with a mix of sequential, branch, loop, and parallel structures [46], for ease of presentation, we focus only on the sequential structure in this case study. It is a common structure to represent the critical path of an application workflow, thus determining end-to-end application performance. But it is worth noting that our online QoS prediction approach can be used for applications with other compositional structures as well. Fig. 15 depicts a prototype application for flight ordering with a simplified workflow composed of five abstract tasks. Specifically, a customer begins with flight search (s_1) by sending a query to flight suppliers for available flight information. Then the user can book a flight (s_2), while the request is sent to the corresponding supplier for flight reservation. Next, the customer edits necessary passenger information (s_3), and sends an order confirmation request to the supplier (s_4). After confirmation, a payment transaction via online payment services (*e.g.*, Paypal) will be launched (s_5). Upon the payment, the flight purchase is completed.

For evaluation, we randomly choose 10 candidate Web services for each abstract task from our dataset. We ignore the potential issue of functional mismatch here, because it will not make a difference to QoS evaluation. We consider four application execution settings: 1) *Static*. Each task chooses the best service according to inital QoS values, and keeps fixed during the whole execution period. 2) *Random*. Each task picks three best-performing candidate services according to inital QoS values, and randomly replace the slowest one with another when QoS degradation occurs. 3) *Dynamic*. We apply AMF to online QoS prediction and use the prediction results (at matrix density = 10% in Section 5.3) to support runtime service adaptation. That is, each task dynamically chooses the best service according to current QoS predictions. 4) *Optimal*. Assuming all QoS values are known a prior, each task selects the best service accordingly at runtime. This is the optimal case of runtime service adaptaton. Fig. 16(a) presents the representative result of application response time under different execution settings. In the figure, assuming the maximal acceptable response time is $1s$, the *static* execution leads to many times of SLA violations due to QoS fluctuations over time. Meanwhile, two SLA violations happen in the *random* setting. In contrast, we achieve near-optimal application performance with runtime service adaptaton in our *dynamic* setting, reducing the 95th percentile response time from 2.3s (*static*) and 0.8s (*random*) to 0.4s. To avoid bias in randomization, we repeat the experiments for 100 times and report the average results (with standard deviations) on response time as shown in Fig. 16(b). We note that our runtime service adaptation (i.e., *dynamic*) has made a 38% improvement over the *static* setting and a 29% improvement over the *random* setting. The small gap between *dynamic* and *optimal*, however, is due to the inaccuracy in QoS prediction. These results indicate that our online QoS prediction approach can greatly facilitate successful runtime service adaptation.

7 Discussion

In this section, we discuss the potential limitations of this work and provide some directions for future work.

QoS measurement v.s. QoS prediction. One may argue that active measurement (e.g., through invoking a service periodically) is a straightforward way to determine QoS of a service, since it is accurate and easy to implement. For

example, the popular distributed document management database, MongoDB, supports replica selection with active measurement [13]. In this case, however, typically only three replicas exist. When making active measurements at large scale, the measurement overhead would become a main concern. It is infeasible, for example, to ping each node in a large CDN (potentially with thousands of nodes) to identify the best one to serve a user request, because this would incur prohibitive network overhead. Furthermore, many services are not free, which will lead to additional cost of service invocations. In our case study, active measurement needs a cost of 3200 (i.e., 5*10*64) times of service invocations. Our QoS prediction approach, in contrast, reduces this cost by 90% when using prediction results at the matrix density of 10%. This indicates a tradeoff between accuracy and measurement cost. When services become available at a large scale, QoS prediction will become a better fit.

Large-scale QoS collection. A QoS collection framework capable of assembling QoS records from users in real time is needed to support online QoS prediction and subsequent service adaptation decisions. Our work is developed based on underlying QoS collection, but we focus primarily on processing and prediction of QoS values. We have previously developed a WSRec framework [49] for collaborative QoS collection, where a set of QoS collector agents were developed using Apache Axis and further deployed on the global PlanetLab platform to collect QoS records of publicly-available Web services at runtime. Section 3 provides a possible extension of this framework to mitigate the case that some users may not want to reveal their QoS data. In addition, a privacy-preserving scheme is explored in [52] by applying differential privacy techniques to dealing with potential privacy issues of QoS collection from different users. We also expect that a streaming data platform, e.g., Amaon Kinesis[4], is used to collect real-time QoS streams, but we leave such an end-to-end integration of QoS collection and prediction for our future work.

Issues in service adaptation. In practice, there are many issues to handle in order to fully support runtime service adaptation, such as service discovery, QoS monitoring and prediction, service interface mediation, adaptation decision making, workflow reconfiguration, load balancing, etc. Each aspect is a complex issue that deserves deep exploration. For instance, service adaptation naturally brings up the problem of how differences in service interfaces can be resolved at runtime. Many studies (e.g., [34], [38]) have been devoted to developing mediation adapters that can transparently map abstract tasks to different target services. Meanwhile, our work has not considered the congestion of load from different users. In practice, when many users believe that a service has good QoS, they will quickly change to this service, which may thus overload the service. Load balancing is desired to mitigate this issue. The whole design space, however, is obviously larger than that can be explored in a single paper. Therefore, we do not aim to address all these issues but focus only on the QoS prediction problem in this paper, which is important yet less explored before.

Potential improvements in QoS prediction. The current implementation of AMF works as a black-box approach to capturing the inherent characteristics of historical QoS data. But it is potential to further combine temporal information of QoS for better modeling, for example, by using smoothing techniques (e.g., exponential moving average [1]) to reduce the weights of older QoS records or applying time series techniques for change point detection. Furthermore, it is possible to incorporate some other contextual information. For example, users can be profiled and grouped by network location, routing domain, region, etc., because users within the same network more likely experience similar QoS. These more sophisticated QoS models deserve further exploration for improvements in prediction accuracy. Besides, the current evaluations are all conducted offline on our QoS dataset. It is desirable to carry out practical online evaluations on real large-scale applications in future.

4. https://aws.amazon.com/kinesis

8 Related Work

Runtime Service Adaptation. To achieve the goal of runtime service adaptation, a large body of research work has been conducted. Specifically, some work (e.g., [12], [35]) extends BPEL execution engines (e.g., Apache ODE) with an interception and adaptation layer to enable monitoring and recovery of services. Some other work (e.g., [14], [36]) investigates feasible adaptation policies, such as replacing the component services or re-structuring the workflows, to optimize service adaptation actions. Moreover, a service manager (e.g., [16]) is employed to discover all available candidate services that match a specific need, while a QoS manager (e.g., [49]) is used to monitor current QoS values of service invocations and obtain necessary QoS prediction results. Different from most of these studies that focus on adaptation mechanism design, our work aims to address the challenge of online QoS prediction, which is also deemed critical to effective runtime service adaptation as noted in [31].

QoS Prediction. Accurate QoS information is essential for QoS-driven runtime service adaptation. Online testing, as presented in [9], [32], is a straightforward way that actively invokes services for QoS measurement. But this approach is costly due to the additonal service invocations incurred, especially when applied to a large number of candidate services. In recent literature, online prediction approaches [10], [45], which analyze historical QoS data using time series models, have been proposed to detect potential service failures and QoS deviations of working services. As for candidate services, Jiang et al. [23] propose a way of sampling to reduce the cost of continously invoking candidate services. Some other work proposes to facilitate Web service recommendation with QoS prediction by leveraging techniques such as collaborative filtering [49], matrix factorization [50], tensor factorization [47], [48]. However, as we show in our experiments, these approaches are insufficient, either in accuracy or in efficiency, to meet the requirements posed by runtime service adaptation. Our work is motivated to address these limitations.

Collaborative Filtering. Recently, CF has been introduced as a promising technique for many system engineering tasks, such as service recommendation [17], [30], system reliability prediction [50], and QoS-aware datacenter scheduling [20]. Matrix factorization (MF) is one of the most

widely-used models in collaborative filtering, which has recently inspired many applications in other domains. In our previous work [53], we apply the MF model, along with efficient shortest distance computation, to guide dynamic request routing for large-scale cloud applications. Some other work (e.g., [30], [50]) introduces MF as a promising model for Web service recommendation. However, such direct uses of the MF model, as we show in Section 5, are insufficient in addressing the challenges in online QoS prediction of runtime service adaptation. Therefore, our work is aimed to extend the MF model in terms of accuracy, efficiency, and robustness. The weighted matrix factorization has been once studied in [42], but our approach differs from it in that we apply adaptive weights instead of fixed weights in the iteration process to maintain robustness of the model.

Online Learning. Online learning algorithms [3] are commonly used for training large-scale datasets where traditional batch algorithms are computationally infeasible, or in situations where it is necessary to dynamically adapt the models with the sequential arrival of data. Recently, the use of online learning in collaborative filtering has received emerging attention. In an early study [18], Google researchers solve the online learning problem of neighbourhood-based collaborative filtering for google news recommendation. The following studies [27], [29] apply stochastic gradient descent, a randomized version of batch-mode gradient descent, to optimizing the matrix factorization based formulation of collaborative filtering in an online fashion. Some other work (see the survey [24]) further explores the use of parallel computing to speed up the algorithms. Inspired by these successful applications, we propose adaptive online learning, by customizing the widely-used stochastic gradient descent algorithm with adaptive learning weights, to guarantee robust, online QoS prediction.

9 Conclusion

This is the first work targeting on predicting QoS of candidate services for runtime service adaptation. Towards this end, we propose adaptive matrix factorization (AMF), an online QoS prediction approach. AMF formulates QoS prediction as a collaborative filtering problem inspired from recommender systems, and significantly extends the traditional matrix factorization model with techniques of data transformation, online learning, and adaptive weights. The evaluation results, together with a case study, on a real-world QoS dataset of Web services have validated AMF in terms of accuracy, efficiency, and robustness.

Acknowledgment

The work described in this paper was supported by the National Natural Science Foundation of China (Project No. 61332010 and 61472338), the National Basic Research Program of China (973 Project No. 2014CB347701), the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 415113 of the General Research Fund), and 2015 Microsoft Research Asia Collaborative Research Program (Project No. FY16-RESTHEME-005). Pinjia He and Zibin Zheng are the corresponding authors.

References

[1] Exponential moving average. http://en.wikipedia.org/wiki/Moving_average. [Accessed: 15-May-2016].
[2] Gradient descent. http://en.wikipedia.org/wiki/Gradient_descent. [Accessed: 15-May-2016].
[3] Online learning. https://en.wikipedia.org/wiki/Online_machine_learning. [Accessed: 15-May-2016].
[4] scipy.stats.boxcox reference guide. http://docs.scipy.org/doc/scipy-0.17.0/reference/generated/scipy.stats.boxcox.html. [Accessed: 15-May-2016].
[5] Singular value decomposition (SVD). http://en.wikipedia.org/wiki/Singular_value_decomposition. [Accessed: 15-May-2016].
[6] The state of API 2016: Growth, opportunities, challenges, & processes for the API industry. http://blog.smartbear.com/api-testing/api-trends-2016/. [Accessed: 10-Jan-2017].
[7] Web service. https://en.wikipedia.org/wiki/Web_service. [Accessed: 5-Apr-2016].
[8] WS-DREAM: Towards open datasets and source code for web service research. http://wsdream.github.io.
[9] M. Ali, F. D. Angelis, D. Fanì, A. Bertolino, G. D. Angelis, and A. Polini. An extensible framework for online testing of choreographed services. *IEEE Computer*, 47(2):23–29, 2014.
[10] A. Amin, L. Grunske, and A. Colman. An automated approach to forecasting qos attributes based on linear and non-linear time series modeling. In *Proc. of IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 130–139, 2012.
[11] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.
[12] L. Baresi and S. Guinea. Self-supervising bpel processes. *IEEE Trans. Software Eng. (TSE)*, 37(2):247–263, 2011.
[13] K. Bogdanov, M. P. Quirós, G. Q. M. Jr., and D. Kostic. The nearest replica can be farther than you think. In *Proc. of the Sixth ACM Symposium on Cloud Computing (SoCC)*, pages 16–29, 2015.
[14] V. Cardellini, E. Casalicchio, V. Grassi, S. Iannucci, F. L. Presti, and R. Mirandola. MOSES: A framework for QoS driven runtime adaptation of service-oriented systems. *IEEE Trans. Software Eng. (TSE)*, 38(5):1138–1159, 2012.
[15] W. Chen and I. Paik. Toward better quality of service composition based on a global social service network. *IEEE Trans. Parallel Distrib. Syst. (TPDS)*, 26(5):1466–1476, 2015.
[16] W. Chen, I. Paik, and P. C. K. Hung. Constructing a global social service network for better quality of web service discovery. *IEEE Trans. Services Computing (TSC)*, 8(2):284–298, 2015.
[17] X. Chen, Z. Zheng, Q. Yu, and M. R. Lyu. Web service recommendation via exploiting location and qos information. *IEEE Trans. Parallel Distrib. Syst. (TPDS)*, 25(7):1913–1924, 2014.
[18] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proc. of the 16th International Conference on World Wide Web (WWW)*, pages 271–280, 2007.
[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proc. of the 21st ACM Symposium on Operating Systems Principles (SOSP)*, pages 205–220, 2007.
[20] C. Delimitrou and C. Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *Proc. of the ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.
[21] J. Harney and P. Doshi. Speeding up adaptation of web service compositions using expiration times. In *Proc. of the 16th ACM International Conference on World Wide Web (WWW)*, pages 1023–1032, 2007.
[22] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, 2004.
[23] B. Jiang, W. K. Chan, Z. Zhang, and T. H. Tse. Where to adapt dynamic service compositions. In *Proc. of the ACM International Conference on World Wide Web (WWW)*, 2009.
[24] E. Karydi and K. G. Margaritis. Parallel and distributed collaborative filtering: A survey. *CoRR*, abs/1409.2762, 2014.
[25] A. Klein, F. Ishikawa, and S. Honiden. Towards network-aware service composition in the cloud. In *Proc. of the 21st World Wide Web Conference (WWW)*, pages 959–968, 2012.

[26] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[27] Y. Koren. Collaborative filtering with temporal dynamics. *Commun. ACM*, 53(4):89–97, 2010.

[28] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *Proc. of IEEE International Conference on Web Services (ICWS)*, 2010.

[29] G. Ling, H. Yang, I. King, and M. R. Lyu. Online learning for collaborative filtering. In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2012.

[30] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu. An extended matrix factorization approach for qos prediction in service selection. In *Proc. of IEEE Ninth International Conference on Services Computing (SCC)*, 2012.

[31] A. Metzger, C.-H. Chi, Y. Engel, and A. Marconi. Research challenges on online service quality prediction for proactive adaptation. In *Proc. of the Workshop on European Software Services and Systems Research - Results and Challenges*, pages 51–57, 2012.

[32] A. Metzger, O. Sammodi, and K. Pohl. Accurate proactive adaptation of service-oriented systems. In *Proc. of Assurances for Self-Adaptive Systems - Principles, Models, and Techniques*. 2013.

[33] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka. Towards proactive adaptation with confidence: augmenting service monitoring with online testing. In *Proc. of ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*, pages 20–28, 2010.

[34] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-end support for qos-aware service selection, binding, and mediation in vresco. *IEEE Trans. Services Computing (TSC)*, 3(3):193–205, 2010.

[35] O. Moser, F. Rosenberg, and S. Dustdar. Non-intrusive monitoring and service adaptation for ws-bpel. In *Proc. of ACM International Conference on World Wide Web (WWW)*, 2008.

[36] V. Nallur and R. Bahsoon. A decentralized self-adaptation mechanism for service-based applications in the cloud. *IEEE Trans. Software Eng. (TSE)*, 39(5):591–612, 2013.

[37] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-oriented computing: State of the art and research challenges. *IEEE Computer*, 40(11):38–45, 2007.

[38] S. D. Philipp Leitner, Anton Michlmayr. Towards flexible interface mediation for dynamic service invocations. In *Proc. of the 3rd Workshop on Emerging Web Services Technology*, 2008.

[39] R. M. Sakia. The box-cox transformation technique: A review. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 41(2):169–178, 1992.

[40] R. Salakhutdinov and A. Mnih. Probabilistic matrix factorization. In *Proc. of the Twenty-First Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

[41] A. Shapiro and Y. Wardi. Convergence analysis of gradient descent stochastic algorithms. *Journal of Optimization Theory and Applications*, pages 45–4, 1996.

[42] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *Proc. of International Conference on Machine Learning (ICML)*, pages 720–727, 2003.

[43] D. Strom and J. F. van der Zwet. Truth and lies about latency in the cloud. Interxion™ white paper, 2012.

[44] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. Artificial Intellegence*, 2009.

[45] C. Wang and J.-L. Pazat. A two-phase online prediction approach for accurate and timely adaptation decision. In *Proc. of IEEE International Conference on Services Computing (SCC)*, pages 218–225, 2012.

[46] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Trans. Software Eng. (TSE)*, 30(5):311–327, 2004.

[47] W. Zhang, H. Sun, X. Liu, and X. Guo. Temporal qos-aware web service recommendation via non-negative tensor factorization. In *Proc. of the 23rd International World Wide Web Conference (WWW)*, pages 585–596, 2014.

[48] Y. Zhang, Z. Zheng, and M. R. Lyu. Wspred: A time-aware personalized qos prediction framework for web services. In *Proc. of the 22nd IEEE International Symposium on Software Reliability Engineering (ISSRE)*, pages 210–219, 2011.

[49] Z. Zheng, H. Ma, M. R. Lyu, and I. King. WSRec: A collaborative filtering based web service recommender system. In *Proc. of IEEE International Conference on Web Services (ICWS)*, 2009.

[50] Z. Zheng, H. Ma, M. R. Lyu, and I. King. Collaborative web service qos prediction via neighborhood integrated matrix factorization. *IEEE T. Services Computing (TSC)*, 6(3):289–299, 2013.

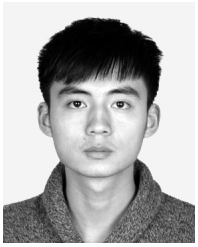
[51] J. Zhu, P. He, Z. Zheng, and M. R. Lyu. Towards online, accurate, and scalable qos prediction for runtime service adaptation. In *Proc. of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 318–327, 2014.

[52] J. Zhu, P. He, Z. Zheng, and M. R. Lyu. A privacy-preserving qos prediction framework for web service recommendation. In *Proc. of the IEEE International Conference on Web Services (ICWS)*, pages 241–248, 2015.

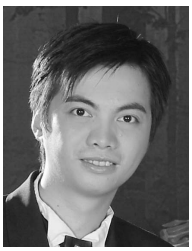
[53] J. Zhu, Z. Zheng, and M. R. Lyu. DR^2: Dynamic request routing for tolerating latency variability in online cloud applications. In *Proc. of IEEE International Conference on Cloud Computing (CLOUD)*, pages 589–596, 2013.

[54] J. Zhu, Z. Zheng, Y. Zhou, and M. R. Lyu. Scaling service-oriented applications into geo-distributed clouds. In *Proc. of IEEE International Symposium on Service-Oriented System Engineering (SOSE)*, pages 335–340, 2013.

Jieming Zhu is currently a postdoctoral fellow at The Chinese University of Hong Kong. He received the B.Eng. degree in information engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2011; the Ph.D. degree from Department of Computer Science and Engineering, The Chinese University of Hong Kong, in 2016. His current research focuses on data analytics and its applications in software reliability engineering.



Pinjia He is currently a Ph.D. student from Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China. He received the B.Eng. degree in computer science and technology from South China University of Technology, Guangzhou, China, in 2013. His current research interests include log analysis, system reliability, software engineering, and distributed system.



Zibin Zheng is currently an associate professor at Sun Yat-sen University, China. He received his Ph.D. degree from Department of Computer Science and Engineering, The Chinese University of Hong Kong, in 2011. He received ACM SIGSOFT Distinguished Paper Award at ICSE2010, Best Student Paper Award at ICWS2010, and IBM Ph.D. Fellowship Award 2010-2011. His research interests include services computing, software engineering and block chain.

Michael R. Lyu is currently a professor of Department of Computer Science and Engineering, The Chinese University of Hong Kong. He received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1981; the M.S. degree in computer engineering from University of California, Santa Barbara, in 1985; and the Ph.D. degree in computer science from the University of California, Los Angeles, in 1988. His research interests include software reliability engineering, distributed systems, fault-tolerant computing, and machine learning. Dr. Lyu is an ACM Fellow, an IEEE Fellow, an AAAS Fellow, and a Croucher Senior Research Fellow for his contributions to software reliability engineering and software fault tolerance.