Incorporating Invocation Time in Predicting Web Service QoS via Triadic Factorization

Wancai Zhang, Hailong Sun, Xudong Liu, Xiaohui Guo
School of Computer Science and Engineering, Beihang University
Beijing, China
{zhangwc, sunhl, liuxd, guoxh}@act.buaa.edu.cn

***Abstract*—With the development of Service-Oriented technologies, the amount of Web services grows rapidly. QoS-Aware Web service recommendation can help service users to design more efficient service-oriented systems. However, existing methods assume the QoS information for service users are all known and accurate, but in real case, there are always many missing QoS values in history records, which increase the difficulty of the missing QoS value prediction. By considering the *user-service-time* three dimension context information, we study a Temporal QoS-Aware Web Service Prediction Framework which aims to recommend best candidates to service user's requirements and meanwhile improve the QoS prediction accuracy. One major challenge is that how to deal with the high dimension, sparse QoS value data. Tensor which is known as multi-way array provides a natural representation for such QoS value data. Therefore, we formalize this problem as a tensor factorization model and propose a Tucker Decomposition (TD) algorithm which is able to deal with the triadic relations of *user-service-time* model. Extensive experiments are conducted based on our real-world QoS dataset collected on Planet-Lab, comprised of service invocation response-time values from 408 users on 5,473 Web services at 56 time periods. Comprehensive empirical studies demonstrate that our approach is more accuracy than other approaches and achieves $100X$ to $1000X$ memory space reduction.**

***Keywords*-Web service recommendation; Tensor factorization; Collaborative Filtering; QoS Prediction;**

I. Introduction

Web services are considered as software functions provided at network addresses over the web or the cloud. They encapsulate application functionality and make them available through interfaces. In recent years, how to provide Web service recommendation that seamlessly satisfy service users' needs anywhere and anytime becomes a challenging research problem in Web service recommendation research area.

Service recommendation researches mainly focus on the consideration of the services' non-functional properties. The emergence of multiple Web services with the same functionality offers a set of alternatives for users to pick over based on their *Quality of Service* (QoS). QoS is a broad concept that encompasses a number of properties such as price, availability, and reputation [1].

With the growing number of functionally equivalent services on the web, it is quite important to recommend services considering their non-functional QoS properties. The high QoS of Web services can help service users to reduce re-engineering cost and to produce more effective service-oriented systems. However, the QoS performance of Web services observed from the user is usually quite different from that declared by the service providers due to the following reasons:

- Invocations of Web services may be charged since Web service are owned and hosted by different organizations. Furthermore, Web service invocation may produce irreversible effect in the real world.
- There are too many service candidates to be evaluated so that even if the invocations are free, executing a large number of Web service invocations consumes resources of both the service users and the service providers.
- QoS performance of Web services is highly related to user location and the invocation time, since the service status and the network environment (e.g., workload, number of clients, etc.) change over time.

In reality, service users usually only invoke a limited number of Web services and only observe the QoS value at these invocation time. So we should be able to predict the missing QoS value of Web services from distribution location users at different invocation time to optimize the Web services recommendation.

Based on the above analysis, we propose a Temporal QoS-Aware Web Service Prediction Framework, which employs an novel Collaborative Filtering (CF) algorithm for Web service recommendation. The CF method can predict the missing QoS value of Web services by employing their historical QoS values. But there are several challenges to be addressed when applying the temporal QoS-aware prediction: 1) How to collect the QoS information of Web services from distribution location service users? 2) How to improve the CF method to adopt to our temporal QoS-aware prediction? 3) How to deal with the large, sparse and high-dimensional QoS value data?

The most straightforward way for obtaining user-observed Web service QoS value is to conduct evaluation on all the service candidates. Unfortunately, traditional exhaustive testing becomes time-consuming, expensive, and sometimes even impossible to make a comprehensive evaluation of all Web service candidates. To address this challenge, we design a client-side middleware for service users to automatically record QoS value of invocations and contribute the local records to the server. Furthermore, the QoS value of Web services invoked by service users can be represented by two-dimensional *user-service* matrix at each invocation time. Due

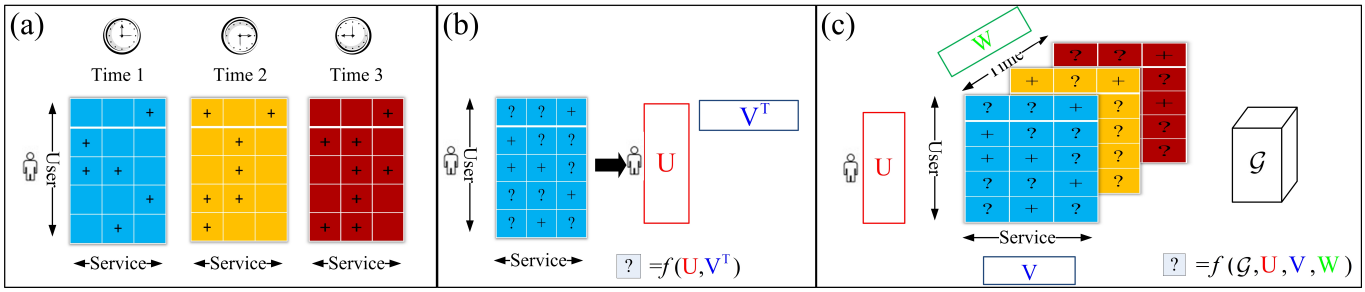

Fig. 1. (a) The Web service QoS value matrix is sparse, so the missing value phenomenon is ubiquitous; (b) The other model-based CF method for QoS prediction; (c) Our temporal QoS-aware triadic factorization for Web Service prediction.

to there are lots of missing QoS values, the matrix is sparse as shown in Figure 1 (a). Some previous works [2], [3], [4], [5], [6] predict the missing values with the CF method. The key idea of the CF method depicted in Figure 1 (b) is the low-rank Matrix Factorization (MF) model [7] which characterizes both users and services by vectors of factors in a joint latent factor space of low dimension. We aim to improve the QoS prediction by using the temporal information, so the prediction method should take into consideration the triadic relation *user-service-time*. To learn such triadic factors, intuitively a tensor factorization(TF) [8] could be introduced. We propose the Tucker model [9] to represent the three-dimensional QoS value tensor. As illustrated in Figure 1 (c), the Tucker decomposition (TD) algorithm attempts to make the temporal QoS-aware prediction for the missing values. Hence each QoS value can be viewed as the result of triadic interaction among user, service and time factors.

In summary, this paper makes the following contributions:

- We extend the two-dimensional *user-service* matrix into a more complicated *user-service-time* triadic relations represented by three-dimensional tensor, and present a novel TF approach that is based on a generalization of MF.
- There are lots of missing QoS values in the *user-service-time* tensor where the majority of the entries are zero. Since the QoS value data may be too large to be stored in memory, we employ two methods to efficient store these values.
- We evaluate our approach experimentally using real-world Web service dataset, which contains more than 37 millions real-world Web service invocation results from 408 distributed service users on 5,473 real-world Web services at 56 time periods. Our real-world Web service dataset has been published online[1].

II. Preliminaries

Before clarifying our model, some basic notations and operations for tensor are first introduced, which are necessary for further understanding Tucker Decomposition.

[1]http://www.service4all.org.cn/servicexchange/

A. Tensor Notation

Definition 1. ***(Tensor)*** *A tensor is a multi-way array. The order of a tensor is the number of dimensions, also known as ways or modes.*

In this paper, tensors are denoted by bold calligraphic upper-case letters $\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{B}} \cdots$, matrices (tensors of order two) by bold upper-case letters $\mathbf{A}$,$\mathbf{B} \cdots$, vectors (tensors of order one) by bold lower-case letters $\mathbf{a}$,$\mathbf{b} \cdots$ and scalars (tensors of order zero) by lower-case letters $a, b \cdots$. An N-way tensor is denoted as: $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, which has N indices $(i_1, i_2, \cdots, i_N)$ and its elements are denoted by $a_{i_1 i_2 \cdots i_N}$.

B. Tensor Operations

Definition 2. ***(Matricization)*** *Matricization also known as unfolding or flattening, is the process of reordering the elements of an N-way array into a matrix.*

The n-mode matricization operation maps a tensor into a matrix. The n-mode matrix of an N-way tensor $\boldsymbol{\mathcal{A}}$ is the I_n-dimensional matrix obtained from $\boldsymbol{\mathcal{A}}$ by varying the index i_n and keeping the other indices fixed, and the elements of $\boldsymbol{\mathcal{A}}$ is mapped into the unfolding matrix $\mathbf{A}_{(n)} \in \mathbb{R}^{I_n \times (I_1 I_2 \cdots I_{n-1} I_{n+1} \cdots I_N)}$. For example, $\mathbf{A}_{(2)}$ represents the mapping $\boldsymbol{\mathcal{A}}^{I \times J \times K} \rightarrow \mathbf{A}_{(2)}^{J \times IK}$.

Definition 3. ***(Mode-n Matrix Product)*** *The n-mode matrix product of a tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a matrix $\boldsymbol{U} \in \mathbb{R}^{J \times I_n}$ is denoted by $\boldsymbol{\mathcal{A}} \times_n \boldsymbol{U}$ and is of size $I_1 \times \cdots \times I_{n-1} \times J \times I_{n+1} \times \cdots \times I_N$. Element-wise, we have*

$$(\boldsymbol{\mathcal{A}} \times_n \boldsymbol{U})_{i_1 \cdots i_{n-1} j i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} u_{j i_n}. \quad (1)$$

Multiple mode-n matrix products can be performed in any order. Following [10], we use the following shorthand notation for multiplication in every mode:

$$\boldsymbol{\mathcal{A}} \times \{\mathbf{A}\} = \boldsymbol{\mathcal{A}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}.$$

and for multiplication in every mode but one:

$$\boldsymbol{\mathcal{A}} \times_{(-n)} \{\mathbf{A}\} = \boldsymbol{\mathcal{A}} \times_1 \mathbf{A}^{(1)} \cdots \times_{(n-1)} \mathbf{A}^{(n-1)} \times_{(n+1)} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}.$$

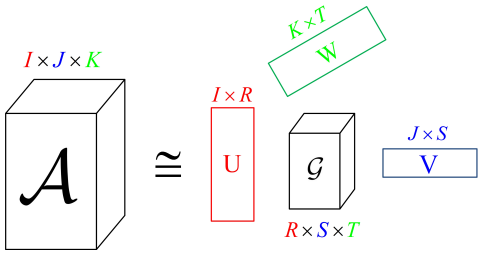

Fig. 2. Tucker decomposition of a 3-way tensor

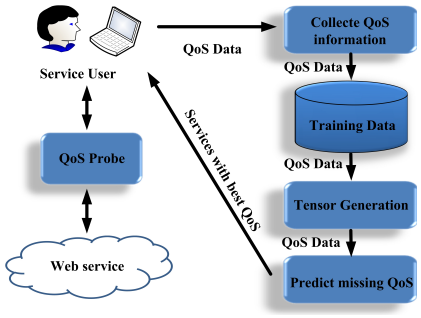

Fig. 3. Temporal QoS-Aware Web Service Prediction Framework

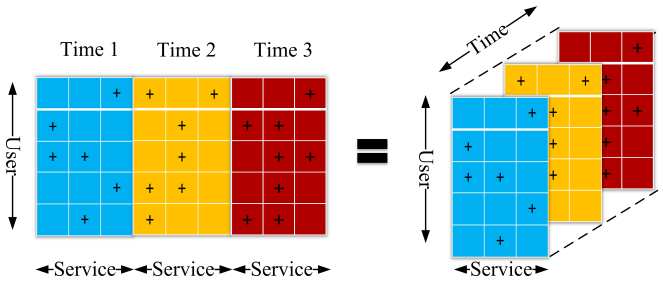

Fig. 4. The observed Web Service QoS value is a ternary relationship that can be seen as a three-dimensional tensor. For each matrix is given that contains the users and services given for a specific time.

Definition 4. ***(Mode-n Vector Product)*** *The n-mode vector product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ with a vector $\boldsymbol{v} \in \mathbb{R}^{I_n}$ is denoted by $\mathcal{A} \bar{\times}_n \boldsymbol{v}$ and is of size $I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N$. Element-wise, we have*

$$(\mathcal{A} \bar{\times}_n \boldsymbol{v})_{i_1 \cdots i_{n-1} i_{n+1} \cdots i_N} = \sum_{i_n=1}^{I_n} x_{i_1 i_2 \cdots i_N} v_{i_n}. \tag{2}$$

C. Tensor Decompositions

To analyze tensor data, many tensor decompositions have been proposed in [8], among which, CANDECOMP/PARAFAC (CP) [11] and Tucker decomposition [9] are the two most popular. Here we focus on the Tucker decomposition. The Tucker decomposition has been applied in many different domains such as web search mining [12] and context recommendation [13].

Definition 5. ***(Tucker Decomposition)*** *Let $\mathcal{A}$ be a tensor of size $I_1 \times I_2 \times \cdots \times I_N$ and $\hat{\mathcal{A}}$ be the approximate result of tensor $\mathcal{A}$. Tucker decomposition of $\mathcal{A}$ yields a core tensor $\mathcal{G}$ of specified size $J_1 \times J_2 \times \cdots \times J_N$ and factor matrices $\boldsymbol{A}^{(n)}$ of size $I_n \times J_n$ for $n = 1, \cdots, N$. Thus, we have*

$$\begin{aligned} \mathcal{A} &\approx \hat{\mathcal{A}} = \mathcal{G} \times_1 \boldsymbol{A}^{(1)} \times_2 \boldsymbol{A}^{(2)} \cdots \times_N \boldsymbol{A}^{(N)} = \mathcal{G} \times \{\boldsymbol{A}\} \\ &= \sum_{i_1=1}^{J_1} \sum_{i_2=1}^{J_2} \cdots \sum_{i_N=1}^{J_N} \mathcal{G}_{i_1 i_2 \cdots i_N} \boldsymbol{a}_{i_1}^{(1)} \circ \boldsymbol{a}_{i_2}^{(2)} \circ \cdots \circ \boldsymbol{a}_{i_N}^{(N)}. \end{aligned} \tag{3}$$

The Tucker decomposition approximates a tensor as a smaller core tensor times the product of factor matrices that span appropriate subspaces in each mode, as illustrated in Figure 2 for $N = 3$.

III. Temporal QoS-Aware Prediction

As presented in the previous section, different users will have different QoS experiences when invoking even the same Web service, due to their various locations and requesting time pionts, and it is expensive to got all their QoS information from user perspective. To address this problem, a user-collaborative mechanism is proposed in our framework for Web service recommendation. The design of user-collaborative mechanism is inspired by the success of *Wikipedia*[2] and *Netflix*[3]. By contributing more past Web service invoking QoS data, the service users can obtain more accurate QoS value prediction of the Web services which they did not use before. We employ the Temporal QoS-Aware Web Service Prediction Framework to make prediction of missing QoS values. As shown in Figure 3, our system collects Web services QoS information from different service users. The more service QoS information contributions, the higher QoS value prediction accuracy can be achieved. After collecting a large number of QoS information, we filter some inferior QoS information for the training data and employ the prediction engine to generate the three-dimensional QoS data for predicting the missing QoS values. Due to the space limitation, we mainly introduce the prediction algorithm principle in this paper.

A. Problem Formulation

When a service user invokes a Web service, the QoS property performance will be collected by our recommendation system. After running a period of time, the system accumulates a collection of Web service QoS value data. It can be represented by a set of quadruplets $\langle$*UserID, ServiceID, TimeID, Value*$\rangle$ (or $\langle u, s, t, v \rangle$ for short), where *UserID, ServiceID, TimeID* is defined as the three-dimensional index according to Each QoS *Value*. Using the QoS value data, a three-dimensional tensor $\mathcal{A} \in \mathbb{R}^{I \times J \times K}$ can be constructed as shown in Figure 4, where I, J and K are the number of *UserID*, *ServiceID* and *TimeID*, respectively. Each entry $\mathcal{A}_{ijk}$ in the tensor represents the QoS value of Web service j observed by the service user i at the invoking time k. $\mathcal{A}_{ijk} = null$, if user i did not invoke Web service j previously. The missing value of Web service is a common phenomenon, even though the density of the dataset collected by our system is only 30%, so

[2]http://www.wikipedia.org
[3]https://signup.netflix.com/global

our QoS value data $\boldsymbol{\mathcal{A}}$ is a large and sparse tensor.

B. Sparse Tensor Representation

Beyond being large data volume, another key characteristic associated with this QoS value data is sparsity, since that most of entries in the tensor are zeros. Therefore, it is much more efficient to store the nonzero elements only. However, the challenge is that given a large sparse tensor, how to efficiently store and analyze it with appropriate memory?

There are lots of options for storing sparse tensors. In this paper we adopt two methods, and the first one is coordinate format which stores both the nonzeros and their indices as proposed in [14]. Assume $\boldsymbol{\mathcal{A}}$ is a sparse tensor of size $I_1 \times I_2 \times \cdots \times I_N$ with $P = nnz(\boldsymbol{\mathcal{A}})$ nonzero values. We store the tensor as

$$\mathbf{v} \in \mathbb{R}^P \quad and \quad \mathbf{S} \in \mathbb{N}^{P \times N}$$

where the pth nonzero value is given by v_p and its subscript is given by the pth row of $\mathbf{S}$, i.e., $\mathbf{s}_{p:}$. In other words,

$$x_{s_{p1}, s_{p2}, \cdots, s_{pN}} = v_p.$$

The total storage for a sparse tensor with P non-zeros is the $(N+1)P$ elements. This is the format implemented in the Matlab Tensor Toolbox [15].

The second method, which storing $\boldsymbol{\mathcal{A}}$ as a Tucker tensor, also has major advantages in term of memory requirements. Let $\boldsymbol{\mathcal{G}}$ be the core tensor of $\boldsymbol{\mathcal{A}}$ as defined in Definition 5. $\boldsymbol{\mathcal{A}}$ requires storage of $\prod_{n=1}^{N} I_n$ elements in its explicit form versus storage of

$$\textbf{STORAGE}(\boldsymbol{\mathcal{G}}) + \sum_{n=1}^{N} I_n J_n \tag{4}$$

elements for the factored form. Thus, the Tucker tensor factored format is clearly advantageous if $\textbf{STORAGE}(\boldsymbol{\mathcal{G}})$ is sufficiently small. Note that, the core tensor $\boldsymbol{\mathcal{G}}$ is not assumed to be dense.

C. Tucker Decomposition Algorithm

We want to find a approximation decomposition as definition in Equation (3) such that the core tensor $\boldsymbol{\mathcal{G}}$ has much smaller dimensions than the original sparse tensor $\boldsymbol{\mathcal{A}}$; in other words, we assume

$$J_n \ll I_n.$$

Thus, the tensor $\boldsymbol{\mathcal{A}}$ which is up to $\prod_{J_1}^{J_N} J_n$ elements can easily fit in memory.

In fitting Tucker decomposition, the goal is to find a core tensor $\boldsymbol{\mathcal{G}}$ and factor matrices $\mathbf{A}^{(n)}$ so that we minimize:

$$\min_{\boldsymbol{\mathcal{G}}, \mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(N)}} \left\| \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{G}} \times \{\mathbf{A}\} \right\|_F^2. \tag{5}$$

where $\| \cdot \|_F$ is the Frobenius norm. The core tensor $\boldsymbol{\mathcal{G}}$ is determined uniquely by the factor matrices. It means that given a fixed set of factor matrices, the optimal result core tensor is

$$\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{B}} \times \{\mathbf{A}\}. \tag{6}$$

The straightforward method of computing the error $e = \left\| \boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{G}} \times \{\mathbf{A}\} \right\|_F^2$ is time-consuming, since $\boldsymbol{\mathcal{A}} - \boldsymbol{\mathcal{G}} \times \{\mathbf{A}\}$ is a large dense tensor. Fortunately, it can be simplified as $e = \|\boldsymbol{\mathcal{A}}\|_F^2 - \|\boldsymbol{\mathcal{G}}\|_F^2$. As $\|\boldsymbol{\mathcal{A}}\|_F^2$ is fixed, Equation (5) can be rewritten as:

$$\max_{\mathbf{A}^{(1)}, \cdots, \mathbf{A}^{(n)}} \left\| \boldsymbol{\mathcal{A}} \times \{\mathbf{A}^T\} \right\|. \tag{7}$$

A common approach for solving Equation (5) is to use an alternating least squares (ALS) method, solving for one factor matrix at a time while holding the others fixed. In other words, we fix all the factor matrices except $\mathbf{A}^{(n)}$ and solve

$$\max_{\mathbf{A}^{(n)}} \left\| \boldsymbol{\mathcal{A}} \times \{\mathbf{A}^T\} \right\|. \tag{8}$$

By setting

$$\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{A}} \times_{-n} \{\mathbf{A}^T\}, \tag{9}$$

we can rewritten Equation (8) as

$$\max_{\mathbf{A}^{(n)}} \left\| {\mathbf{A}^{(n)}}^T \mathbf{B}_{(n)} \right\|, \tag{10}$$

which is solved via the SVD of $\mathbf{B}_{(n)}$ and for more details see [16]. Above the derivation corresponding to ALS algorithm for Tucker decomposition, the pseudo-code of this algorithm is given by Algorithm 1.

Algorithm 1 : The ALS algorithm for Tucker decomposition

Input: the tensor $\boldsymbol{\mathcal{A}}$ of size $I_1 \times I_2 \times \cdots \times I_N$,
the size vector $\mathbf{g}$ of core tensor $\boldsymbol{\mathcal{G}}$ $J_1 \times J_2 \times \cdots \times J_N$.

Output: the approximate tensor $\hat{\boldsymbol{\mathcal{A}}}$,
Core tensor $\boldsymbol{\mathcal{G}}$,
factor matrices $\mathbf{A}^{(n)}$ of size $I_n \times J_n$.

1: **Procedure** $[\hat{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{G}}, \mathbf{A}^{(n)}]$ = TD($\boldsymbol{\mathcal{A}}, \mathbf{g}$).
2: **Initialize**: $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times J_n}$, for $n = 1, \cdots, N$ by small random values.
3: **Repeat**
4: **for** $n = 1, \cdots, N$ **do**
5: $\boldsymbol{\mathcal{B}} = \boldsymbol{\mathcal{A}} \times_{-n} \{\mathbf{A}^T\}$,
6: $\mathbf{A}^{(n)} \leftarrow J_n$ leading left singular vectors of $\mathbf{B}_{(n)}$;
7: **end for**
8: **Until** convergence or the maximum number of iterations is exceeded.
9: $\boldsymbol{\mathcal{G}} = \boldsymbol{\mathcal{B}} \times \{\mathbf{A}\}$.
10: $\hat{\boldsymbol{\mathcal{A}}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)}$.
11: **Return**: $\hat{\boldsymbol{\mathcal{A}}}, \boldsymbol{\mathcal{G}}, \mathbf{A}^{(n)}$.
12: **EndProcedure**

D. Complexity Analysis

Most of the time spent by the algorithm is on the calculating of the leading singular vectors that can be solved by SVD. The time complexity of SVD on a matrix$\mathbf{A} \in \mathbb{R}^{m \times n}$ is $\mathcal{O}(mn\min(m,n))$. Therefore, the total time cost of calculating the leading singular vectors in each iteration is $\mathcal{O}(\sum_{n=1}^{N} I_n J_n \min\{I_n, J_n\})$, which means it is efficient for

low-rank approximation when J_n is small. As J_n increases, the time and memory requirements increase fast, making the algorithm inefficient for large J_n. Because unfolding matrices are long and thin, we utilize a simple strategy that can improve the efficiency. Suppose the SVD of $\mathbf{A}$ is $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{U} \in \mathbb{R}^{m\times k}$ and $\mathbf{V} \in \mathbb{R}^{k\times n}$. To avoid the large value of n, we exploit the fact that the eigensystem of $\mathbf{A}\mathbf{A}^T$ is $\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^T$. Then we only need a subset of the column vectors for the left singular matrix $\mathbf{U}$. Therefore, once we have obtained $\mathbf{U}$ from the eigensystem of $\mathbf{A}\mathbf{A}^T$, it is not necessary to compute $\mathbf{V}$ anymore.

E. Missing Value Prediction

The Web service QoS value data is presented as a three-way tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I\times J\times K}$. Then our Tucker decomposition model can be used to approximate the three-dimensional QoS tensor. After the decomposed components which are a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R\times S\times T}$ and a set of factor matrices: $\mathbf{U} \in \mathbb{R}^{I\times R}, \mathbf{V} \in \mathbb{R}^{J\times S}$ and $\mathbf{W} \in \mathbb{R}^{K\times T}$ are learned, we can predict the QoS value of a given service observed by a user at the specific time interval. For the missing value $\hat{\boldsymbol{\mathcal{A}}}_{ijk}$ is predicted as:

$$\hat{\boldsymbol{\mathcal{A}}}_{ijk} = \sum_{r=1}^{R}\sum_{s=1}^{S}\sum_{t=1}^{T}\boldsymbol{\mathcal{G}}_{rst}\mathbf{u}_r \circ \mathbf{v}_s \circ \mathbf{w}_t. \tag{11}$$

IV. Experiments

In this section, we introduce the experiment dataset, evaluation metrics, and the experiment results. Our experiments aim at addressing the following questions: (1) How does our approach compare with other state-of-the-art methods? (2) How do the tensor density and factor matrices dimensionality influence prediction accuracy? The factor matrices dimensionality determines how many the latent factors which have direct influence on prediction accuracy. (3) How efficient can we solve the sparse tensor storage?

We implement the TD algorithm with Matlab. The experiments are conducted on a Dell PowerEdge T620 machine with two Intel Xeon 2.00GHz processors and 16GB RAM, running Window Server 2008. For each experiment, we repeat it 10 times, and report the mean.

A. Experimental Setup

To verify the prediction accuracy of our proposed approach, we implemented a prototype Web service invoke system *ServiceXChange* and a QoS evaluation middleware *QoSDetecter*. The *ServiceXChange* is a platform, more than 20,000 openly-accessible Web services obtained by crawling Web service information from Internet. Recently we have integrated it with our *Service4All*[4] platform: A Cloud Platform for Service-oriented Software Development. Services with *WSDL* file or address are invoked through SOAP based on the classes generated with java. We employ the distributed computers from Planet-Lab[5] to observe and collect evaluation results of the target Web services to our server. After processing the evaluation results, 408 nodes on Planet-Lab were selected as the Web service users and 5,473 publicly available real-world Web services are monitored by each node continuously. Finally, this experiment dataset is consist of these Web services QoS performances of 7 days from October 15 to 21 of 2013 in 56 time intervals lasting for 3 hours.

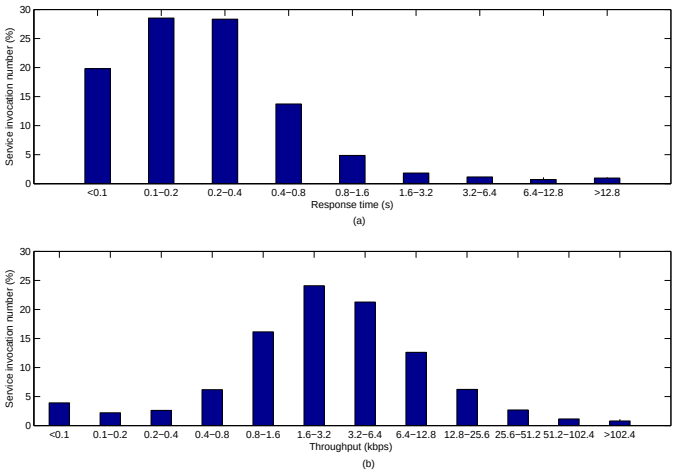

Fig. 5. Distribution of Web Service QoS Value

The dataset contains more than 37 million $\langle u, s, t, v\rangle$ quadruplets, and we obtain two $408 \times 5473 \times 56$ *user-service-time* tensors which contain response time value and throughput value, respectively. Response time is defined as the persistent time between a service user sending a request and receiving the corresponding response, while throughput is defined as the average rate of successful message size per second. The statistics of Web service QoS value dataset are summarized in Table I and the distributions of QoS values are shown in Figure 5. From Table I, the means of response time and throughput is 0.6033 seconds and 8.2107 kbps, respectively. Figure 5 shows that more than 95% of the response time elements are smaller than 1.6 seconds and more than 99% of the throughput elements are smaller than 100 kbps. For simplicity, we use response time and throughput for QoS evaluation in this paper. Without loss of generality, our approach can be easily extended to include more QoS criteria.

TABLE I
Statistics of Web Service QoS Value

Statistics	Response Time	Throughput
Scale of QoS value	0-200s	0-1000kbps
Mean of QoS value	0.6033	8.2107
Num. of service users	408	408
Num. of Web services	5473	5473
Num. of Time periods	56	56

B. Metrics

To evaluate the QoS prediction of accuracy performance, we make use of *Mean Absolute Error* (MAE) and *Root Mean Squared Error* (RMSE) [4] metrics to measure the accuracies and compare them with other commonly used CF approaches.

[4]http://www.service4all.org.cn/
[5]http://www.planet-lab.org/

MAE is defined as:

$$MAE = \frac{1}{N}\sum_{i,j,k}\left|\boldsymbol{\mathcal{A}}_{ijk} - \hat{\boldsymbol{\mathcal{A}}}_{ijk}\right|$$

where $\boldsymbol{\mathcal{A}}_{ijk}$ denotes actual QoS value of Web service j observed by user i at time period k, $\hat{\boldsymbol{\mathcal{A}}}_{ijk}$ represents the predicted QoS value, and N is the number of predicted elements. RMSE is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i,j,k}\left(\boldsymbol{\mathcal{A}}_{ijk} - \hat{\boldsymbol{\mathcal{A}}}_{ijk}\right)^2}$$

C. Baselines

According to the prediction performance measured on the dataset, we investigate whether our method TD can be captured by the following 4 baseline algorithms:

- *UMean*: This method uses the mean QoS value of each user to predict the missing values.
- *IMean*: This method employs the mean QoS value of every service to predict the missing values.
- *WSRec*: This method is a hybrid collaborative algorithm that employs both the similar users and similar Web services for the missing QoS value prediction [3].
- *RSVD*: SVD (Singular Value Decomposition) is proposed by [17] in Collaborative Filtering area, and used to exploit the '*latent structure*' of the original data. We use the regularized SVD method proposed in [18].

D. Accuracy Comparison

In this part, the above 4 baseline methods are compared with TD given the same training and testing cases. Since the baselines cannot be directly applied to 3 dimensions QoS prediction problem, we employ a special formulation for making the comparison with our TD method. We consider the *user-service-time* tensor as a set of *user-service* matrix slices in terms of time interval. Firstly, we compress the tensor into a *user-service* matrix. Each element of this matrix is the average of the specific $\langle user, service\rangle$ pair during all the time intervals. For each slice of the tensor, the baselines are applied for predicting the missing values. Secondly, we compute the MAE and RMSE of baselines, and make the comparison with TD.

Since a service user usually only invokes a very small number of Web services, the dataset is usually very sparse. We randomly remove QoS values to sparse the dataset, and obtain the sparser dataset with different density from 0.01% to 25%. For example, dataset density 1% means that we randomly leave 1% of the dataset for training and the other values become testing set. The parameter settings of TD is that latent features dimensionality set as 30. The comparison result of this experiment are presented in Table II, and the detailed investigations of parameter settings will be provided in the following subsections.

From Table II, we can observe that TD significantly improves the prediction accuracy, and obtains smaller MAE and RMSE values consistently for response time with different matrix densities. With the increase of dataset density from 0.01% to 25%, the MAE and RMSE of TD become smaller, because denser dataset provides more information for the missing value prediction.

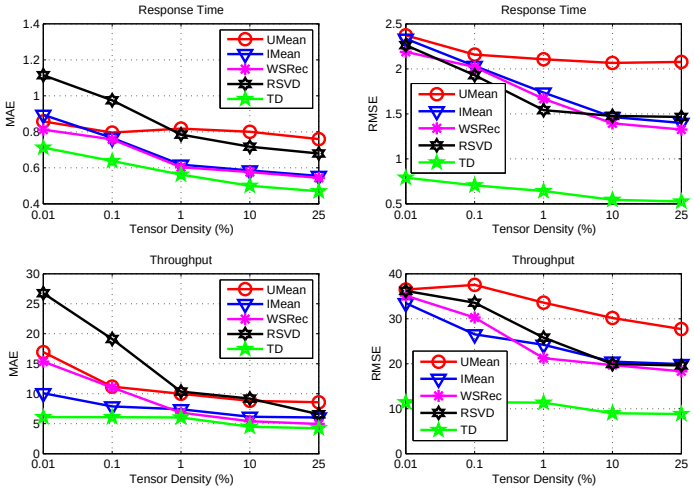

Fig. 6. Impact of Sparseness

1) Impact of Sparseness: We investigate the impact of data sparseness on the prediction accuracy as shown in Figure 6. We set the density of the training tensor as 0.01%, 0.1%, 1%, 10% and 25%. Figure 6 is the MAE and RMSE results of response time and throughput, and it shows that: (1) with the increase of the training density, the performance of TD enhances, indicating that better prediction is achieved with more QoS data. (2) TD outperforms baselines consistently, and the reason is that these baselines only utilize the two-dimensional *user-service* model without considering the more useful triadic relations of user, service and temporal information in the *user-service-time* model. (3) The RMSE of *UMean* is much worse than that of other methods. The reason is that in our dataset the number of users is much smaller than that of services.

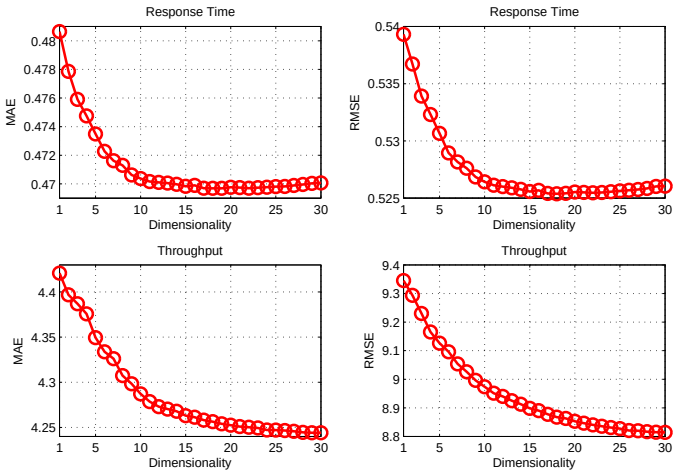

Fig. 7. Impact of Factor Matrices Dimensionality

TABLE II
WEB SERVICE QOS PERFORMANCE COMPARISON (A SMALLER VALUE MEANS A BETTER PERFORMANCE)

WS QoS Property	Methods	MAE					RMSE				
		0.01%	0.1%	1%	10%	25%	0.01%	0.1%	1%	10%	25%
Response Time	UMean	0.8582	0.7959	0.8180	0.8002	0.7597	2.3731	2.1581	2.1072	2.0658	2.0772
	IMean	0.8948	0.7658	0.6176	0.5855	0.5549	2.3344	2.0318	1.7368	1.4643	1.3992
	WSRec	0.8124	0.7582	0.6032	0.5766	0.5440	2.1889	2.0207	1.6667	1.3947	1.3258
	RSVD	1.1145	0.9756	0.7844	0.7177	0.6799	2.2637	1.9291	1.5413	1.4763	1.4656
	TD	**0.7125**	**0.6378**	**0.5620**	**0.5003**	**0.4696**	**0.7905**	**0.7045**	**0.6422**	**0.5449**	**0.5254**
Throughput	UMean	16.942	11.174	9.9737	8.8159	8.5739	36.481	37.530	33.580	30.180	27.719
	IMean	10.087	7.8843	7.4023	6.1553	6.0361	33.460	26.535	24.219	20.465	19.912
	WSRec	15.353	11.035	6.8363	5.4165	4.9443	35.136	30.264	21.221	19.699	18.313
	RSVD	26.759	19.120	10.356	9.1818	6.5823	36.187	33.545	25.818	19.954	19.603
	TD	**6.0956**	**6.0905**	**6.0378**	**4.5001**	**4.2441**	**11.432**	**11.426**	**11.347**	**8.9872**	**8.8166**

2) Impact of Dimensionality: The parameter dimensionality determines how many latent factors involve to tensor factorization. We investigate the impact of the dimensionality by setting the tensor density as 25% and varying the value of dimensionality from 1 to 30 with a step value of one. Figure 7 is the MAE and RMSE results of response time and throughput, and it shows that with the increase of factor matrix dimensionality, the MAE and RMSE keep a declining trend. These observed results coincide with the intuition that relative larger number of latent factor produce smaller error ratio. But, more factors will require longer computation time and storage space. Moreover, when the dimensionality exceeds a certain threshold, it causes the over-fitting problem, which will degrade the prediction accuracy. In this experiment, the threshold of factors is around 20 for response time.

E. Efficiency Comparison

To acquire deeper insights into TD, we present how the space consumption and computational time change when setting different densities of the training tensor as 0.01%, 0.1%, 1%, 10% and 25%. We give a group of experiment results in Figure 8, that shows how different tensor properties affect the memory and running speed when varying each density while setting the dimensionality of TD as 30. Since the data storage format of the four baselines is explicit, one bar named *Baselines* can represent them. *Coordinate* and *Tucker* represent storage of coordinate format and Tucker format, respectively. Compared to *Baselines*, *Coordinate* achieves $1X$ to $1000X$ space reduction, and *Tucker* achieves $100X$ space reduction. As the density drops, the gap between *Coordinate* and *Baselines* becomes bigger as illustrated in Figure 8 (a). But *Tucker* only requires constant memory space, because the Tucker format is decided by Equation 4 which doesn't change with the tensor density. *Tucker* is better than *Coordinate* when the density is moderation, and *Coordinate* is suited to very sparse data. Meanwhile time cost is comparable, we investigate the impact of data sparseness in Figure 8 (b). TD also has a good performance where we set the maximum number of iterations to 100 and for the tolerance on difference in fit to 10^{-4}. Their running times keep increase with the dataset density consistently. Running time is defined as the training time of the algorithm model. Since the both methods *UMean* and *IMean* are simply compute the mean of QoS value, we only compare TD with both *WSRec* and *RSVD*.

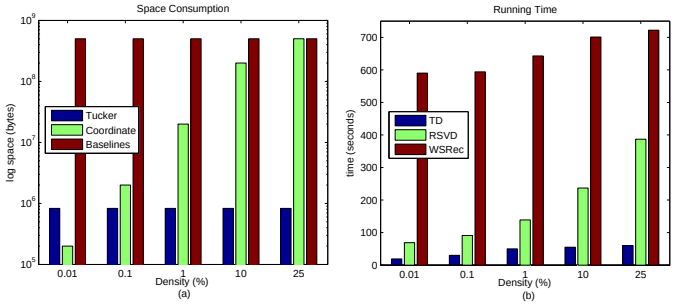

Fig. 8. Efficiency Comparison

V. RELATED WORK

In this section, we briefly introduce the related work in Web service recommendation area. There is a line of work focused on the QoS optimization and Web service recommendation methods, aiming to improve the customers satisfaction. Zeng et al. first transferred the QoS-based service selection problem into an optimization problem in [1]. They used linear programming techniques to find the optimal selection of component services. Alrifai et al. combined the global optimization with local selection in order to find an approximate optimal selection in [19]. After that, they proposed a new method to select and use representative skyline services for QoS-aware Web service composition in [20].

To get personalized QoS, a few works attempted to predict the missing values with applying collaborative filtering methods. Shao et al. [2] first employed CF technique to predict the QoS of Web service. They proposed a user-based CF algorithm to predict QoS value. Zheng et al. [3] proposed a hybrid user-based and item-based CF algorithm to predict the personalized QoS, and they carried out a series of large-scale experiments based on real Web services dataset. Jiang et al. [21] proposed that the influence of personalization of Web service items should be taken into account when computing degree of similarity between users. It is that more popular services or services with more stable QoS from user to user should contribute less to user similarity measurement. Zhang et al. [22] suggested that it was better to combine users' QoS

experiences, environment factor and user input factor to predict QoS value. Chen et al. [6] discovered the great influence of user's location to the accuracy of prediction and proposed a region based hybrid Collaborative Filter algorithm to predict the QoS of services. Lo et al. [5] propose an extended Matrix Factorization (EMF) framework with relational regularization to make missing QoS value prediction. They added a user-based regularization term and a service-based regularization term in the minimizing objective function.

Our work is quite different from these approaches which deal with the *user-service* two-dimensional matrix data without considering of the temporal information of Web service invocation. The absence of temporal information leads QoS prediction to suffering from the static model issue. We argue that in QoS-aware Web service recommendation, the prediction model should consider using the temporal information to reveal the triadic relations of *user-service-time*, rather than the dyadic relations of *user-service*. To learn the triadic relations from the QoS value data, a tensor factorization approach is proposed. The Tucker tensor model [9] is utilized to represent the triadic relations among user, service and time. In this paper, we deal with efficiently storing a sparse three-dimensional tensor whose elements are the QoS value according to the triplet $\langle user, service, time \rangle$ and proposing a TD method, to approximate this QoS value tensor.

VI. Conclusion

In this paper, we extend the Matrix Factorization model to three dimensions through the use of tensor, and employ the tensor factorization approach to advance the QoS-aware Web service prediction performance in considering of temporal information. The TD algorithm incorporates both the specially structured tensors allow for efficient storage and computation and the clustered representation of real-world QoS data. A systematic mechanism for Web service QoS value collection is designed and real-world experiments are conducted. In the experimental results, a higher accuracy of QoS value prediction is obtained with using the three-dimensional *user-service-time* model, when comparing our method with other baseline methods.

In our future work, more real-world Web services will be monitored, and more QoS properties and contextual information will be collected. Besides the temporal contextual information, more contextual information that influences the client-side QoS performance should be considered to improve the prediction accuracy. Since there are so many prediction methods, we also consider to explore an ensemble method for aggregating various algorithms to predict missing QoS value.

Acknowledgment

This work was supported by National Natural Science Foundation of China (No. 61103031, No. 61370057), China 863 program (No. 2012AA011203), China 973 program (No. 2014CB340300, No. 2014CB340304), A Foundation for the Author of National Excellent Doctoral Dissertation of PR China (No. 201159), and Beijing Nova Program(2011022). We are pleased to acknowledge Mingyue Liang for collecting and preparing the dataset.

References

[1] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng, "Quality driven web services composition," in *Proceedings of the 12th international conference on World Wide Web*. ACM, 2003, pp. 411–421.

[2] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei, "Personalized qos prediction forweb services via collaborative filtering," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*. IEEE, 2007, pp. 439–446.

[3] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "Wsrec: A collaborative filtering based web service recommender system," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 437–444.

[4] Z. Zheng, H. Ma, M. Lyu, and I. King, "Collaborative web service qos prediction via neighborhood integrated matrix factorization," 2012.

[5] W. Lo, J. Yin, S. Deng, Y. Li, and Z. Wu, "Collaborative web service qos prediction with location-based regularization," in *Web Services (ICWS), 2012 IEEE 19th International Conference on*. IEEE, 2012, pp. 464–471.

[6] X. Chen, X. Liu, Z. Huang, and H. Sun, "Regionknn: A scalable hybrid collaborative filtering algorithm for personalized web service recommendation," in *Web Services (ICWS), 2010 IEEE International Conference on*. IEEE, 2010, pp. 9–16.

[7] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[8] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[9] L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.

[10] B. W. Bader and T. G. Kolda, "Algorithm 862: Matlab tensor classes for fast algorithm prototyping," *ACM Transactions on Mathematical Software (TOMS)*, vol. 32, no. 4, pp. 635–653, 2006.

[11] J. D. Carroll and J.-J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.

[12] J.-T. Sun, H.-J. Zeng, H. Liu, Y. Lu, and Z. Chen, "Cubesvd: a novel approach to personalized web search," in *Proceedings of the 14th international conference on World Wide Web*. ACM, 2005, pp. 382–390.

[13] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver, "Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering," in *Proceedings of the fourth ACM conference on Recommender systems*. ACM, 2010, pp. 79–86.

[14] B. W. Bader and T. G. Kolda, "Efficient matlab computations with sparse and factored tensors," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 205–231, 2007.

[15] B. W. Bader, T. G. Kolda *et al.*, "Matlab tensor toolbox version 2.5," Available online, January 2012. [Online]. Available: http://www.sandia.gov/ tgkolda/TensorToolbox/

[16] T. G. Kolda, *Multilinear operators for higher-order decompositions*. United States. Department of Energy, 2006.

[17] D. Billsus and M. J. Pazzani, "Learning collaborative information filters." in *ICML*, vol. 98, 1998, pp. 46–54.

[18] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, pp. 5–8.

[19] M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient qos-aware service composition," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 881–890.

[20] M. Alrifai, D. Skoutas, and T. Risse, "Selecting skyline services for qos-based web service composition," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 11–20.

[21] Y. Jiang, J. Liu, M. Tang, and X. Liu, "An effective web service recommendation method based on personalized collaborative filtering," in *Web Services (ICWS), 2011 IEEE International Conference on*. IEEE, 2011, pp. 211–218.

[22] Z. Li, Z. Bin, L. Ying, G. Yan, and Z. Zhi-Liang, "A web service qos prediction approach based on collaborative filtering," in *Services Computing Conference (APSCC), 2010 IEEE Asia-Pacific*. IEEE, 2010, pp. 725–731.