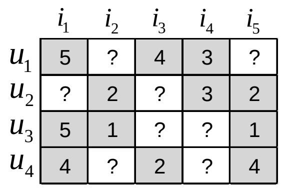
Towards Online, Accurate, and Scalable QoS Prediction for Runtime Service Adaptation [Supplementary Report]

Jieming Zhu, Pinjia He, Zibin Zheng, Michael R. Lyu
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{jmzhu, pjhe, zbzheng, lyu}@cse.cuhk.edu.hk

***Abstract*—This report is provided as supplemental material for our main paper [1]. The content comprises three appendixes as follows: collaborative filtering, mathematical basics, and additional experimental results.**

Appendix A
Collaborative Filtering

Collaborative filtering (CF) techniques are commonly used in commercial recommender systems, such as movie recommendation in Netflix[1] and item recommendation in Amazon[2]. The CF model has been widely studied in recent years. In recommender systems, CF works for the rating prediction problem. Specifically, users likely rate the items that they know about, such as $1 \sim 5$ stars for the moives they have watched or books they have read. As illustrated in Fig. 1, the values in grey entries are observed rating data, and the blank entries are unknown values. For example, the rating value between user u_1 and iterm i_1 is 5, while the rating value between user u_1 and iterm i_5 is missing, because u_1 has not rated i_5. In practice, each user usually rate only a small set out of all of the items, due to the large number of items. As a result, the user-item rating matrix is very sparse.



	i_1	i_2	i_3	i_4	i_5
u_1	5	?	4	3	?
u_2	?	2	?	3	2
u_3	5	1	?	?	1
u_4	4	?	2	?	4

Fig. 1. An Example of Rating Prediction

The basic idea of CF is to exploit and model the observed data to predict the unknown values, based on the insight that similar users may have similar preferences on the same item, and thus have similar ratings. To achieve this goal, two types of CF techniques have been studied in recent literature: neighbourhood-based approaches and model-based approaches [2].

Neighbourhood-based approaches: Neighbourhood-based approaches include user-based approaches (*e.g.*, UPCC) that leverage the similarity between users, item-based approaches (*e.g.*, IPCC) that employ the similarity between items, and their fusions (*e.g.*, UIPCC [3]). However, neighbourhood-based approaches are incapable of handling the data sparsity problem and have high time complexity.

Model-based approaches: Model-based approaches provide a predefined compact model to fit the training data, which can be further used to predict the unknown values. Matrix factorization [4] is one of the most popular model-based approaches used for collaborative filtering. In addition, matrix factorization model can usually achieve better performance than neighbourhood-based approaches.

Appendix B
Mathematical Basics

This section provides some algorithmic background for our adaptive matrix factorization model.

A. Euclidean Norm

Euclidean norm $\|\cdot\|_2$ is a vector norm. Given a vector $V \in \mathbb{R}^n$, its Euclidean norm is defined as follows:

$$\|V\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2} \tag{1}$$

where v_i is the element of V.

B. Frobenius Norm

Frobenius norm $\|\cdot\|_F$ is a matrix norm. Given a matrix $A \in \mathbb{R}^{n \times m}$, its frobenius norm is defined as follows:

$$\|A\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{ij}^2} \tag{2}$$

where a_{ij} is the element of A. When A reduces to a vector, the Frobenius norm is equivalent to the Euclidean norm.

C. Gradient Descent

Gradient descent is a widely used method to find a local minimum of an object function in an iterative way. Note that in our experiments, the approach PMF is implemented by using gradient descent algorithm, as described in the following.

As for matrix factorization model, the object function is given as follows:

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}(R_{ij} - U_i^T S_j)^2 + \frac{\lambda_U}{2}\|U\|_F^2 + \frac{\lambda_S}{2}\|S\|_F^2, \tag{3}$$

[1]http://www.netflix.com
[2]http://www.amazon.com

Algorithm 1: Gradient Descent for MF

```
Input: The collected QoS matrix $R$, the indication matrix $I$,
       and the model parameters: $\eta$, $\lambda_U$ and $\lambda_S$. /* $I_{ij} = 1$
       if $R_{ij}$ is known; otherwise, $I_{ij} = 0$ */
Output: The QoS prediction results: $\hat{R}_{ij}$, where $I_{ij} = 0$.
1  Initialize $U \in \mathbb{R}^{d\times n}$ and $S \in \mathbb{R}^{d\times m}$ randomly;
2  repeat                                  /* Batch-mode updating */
3      foreach (i, j) do        /* Compute $\frac{\partial \mathcal{L}}{\partial U_i}$ and $\frac{\partial \mathcal{L}}{\partial S_j}$ */
4          $\frac{\partial \mathcal{L}}{\partial U_i} \leftarrow \sum_{j=1}^{m} I_{ij}(U_i^T S_j - R_{ij})S_j + \lambda_U U_i$;
5          $\frac{\partial \mathcal{L}}{\partial S_j} \leftarrow \sum_{i=1}^{n} I_{ij}(U_i^T S_j - R_{ij})U_i + \lambda_S S_j$;
6      foreach (i, j) do     /* Update each $U_i$ and $S_j$ */
7          $U_i \leftarrow U_i - \eta\frac{\partial \mathcal{L}}{\partial U_i}$;
8          $S_j \leftarrow S_j - \eta\frac{\partial \mathcal{L}}{\partial S_j}$;
9  until converge;
10 foreach (i, j) $\in \{I_{ij} = 0\}$ do  /* Make prediction */
11     $\hat{R}_{ij} = U_i^T S_j$;
```

where the definition of each symbol has been described in our main paper [1]. Then gradient descent works by updating U_i and S_j simultaneously from random initialization using the following updating rules:

$$U_i \leftarrow U_i - \eta\frac{\partial \mathcal{L}}{\partial U_i}, \quad S_j \leftarrow S_j - \eta\frac{\partial \mathcal{L}}{\partial S_j}, \tag{4}$$

In particular, the derivatives of U_i and S_j can be derived from Equation 3 as follows:

$$\frac{\partial \mathcal{L}}{\partial U_i} = \sum_{j=1}^{m} I_{ij}(U_i^T S_j - R_{ij})S_j + \lambda_U U_i, \tag{5}$$

$$\frac{\partial \mathcal{L}}{\partial S_j} = \sum_{i=1}^{n} I_{ij}(U_i^T S_j - R_{ij})U_i + \lambda_S S_j. \tag{6}$$

Hence, the updating rules in Equation 4 can be rewritten as follows:

$$U_i \leftarrow U_i - \eta\big(\sum_{j=1}^{m} I_{ij}(U_i^T S_j - R_{ij})S_j + \lambda_U U_i\big), \tag{7}$$

$$S_j \leftarrow S_j - \eta\big(\sum_{i=1}^{n} I_{ij}(U_i^T S_j - R_{ij})U_i + \lambda_S S_j\big). \tag{8}$$

Gradient descent works on batch-mode, which needs all the data to be available. The latent factors U_i and S_j move iteratively by a small step of the average gradient, *i.e.*, $\frac{\partial \mathcal{L}}{\partial U_i}$ and $\frac{\partial \mathcal{L}}{\partial S_j}$, where the step size is controlled by η.

The detailed algorithm of gradient descent for MF is presented in Algorithm 1.

D. Stochastic Gradient Descent

The scheme of stochastic gradient descent (SGD) is to update the stochastically using the sequentially coming data. At each step, the model can be adjusted by only considering the current data sample. Thus, SGD naturally provides an online algorithm, where we can adjust the model using each data sample from the data stream in an online fashion.

Algorithm 2: Stochastic Gradient Descent for MF

```
Input: Sequentially observed QoS data samples: $(u_i, s_j, R_{ij})$,
       and the model parameters: $\eta$, $\lambda_u$ and $\lambda_s$.
Output: The QoS prediction results: $\hat{R}_{ij}$, where $I_{ij} = 0$.
1  Initialize $U \in \mathbb{R}^{d\times n}$ and $S \in \mathbb{R}^{d\times m}$ randomly;
2  repeat                         /* Online-mode updating */
3      foreach $(u_i, s_j, R_{ij})$ do
4          $\frac{\partial \ell}{\partial U_i} \leftarrow (U_i^T S_j - R_{ij})S_j + \lambda_u U_i$;
5          $\frac{\partial \ell}{\partial S_j} \leftarrow (U_i^T S_j - R_{ij})U_i + \lambda_s S_j$;
6          $U_i \leftarrow U_i - \eta\frac{\partial \ell}{\partial U_i}$;
7          $S_j \leftarrow S_j - \eta\frac{\partial \ell}{\partial S_j}$;
8  until converge;
9  foreach (i, j) $\in \{I_{ij} = 0\}$ do   /* Make prediction */
10     $\hat{R}_{ij} = U_i^T S_j$;
```

Formally, The loss function $\mathcal{L}$ in Euqation 3 can be seen as the sum of pairwise loss functions:

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{j=1}^{m} I_{ij}\ell(U_i, S_j), \tag{9}$$

and the pairwise loss function $\ell(U_i, S_j)$ with respect to (U_i, S_j, R_ij) is defined as

$$\ell(U_i, S_j) = \frac{1}{2}(R_{ij} - U_i^T S_j)^2 + \frac{\lambda_u}{2}\left\|U_i\right\|_2^2 + \frac{\lambda_s}{2}\left\|S_j\right\|_2^2, \tag{10}$$

Note that the regularization parameters λ_u and λ_s are on different scale from those in Equation 3. Similarly, we can derive the following updating equations for each iteration:

$$U_i \leftarrow U_i - \eta\big((U_i^T S_j - R_{ij})S_j + \lambda_u U_i\big), \tag{11}$$

$$S_j \leftarrow S_j - \eta\big((U_i^T S_j - R_{ij})U_i + \lambda_s S_j\big). \tag{12}$$

The detailed algorithm of stochastic gradient descent for MF is presented in Algorithm 2.

Appendix C
Additional Experimental Results

A. Accuracy Comparison Results

Table I provides the overall accuracy comparison results, which supplements the experimental results shown in our main paper [1]. In particular, some experimental parameters are revised to further optimize our AMF approach. In this experiment, we set $d = 10$, $\beta = 0.3$, $\eta = 0.8$, $\lambda = 0.0003$ for RT, $\lambda = 0.0002$ for TP, and the α is automatically tuned by using the *boxcox* function in Matlab. At each time slice, each approach is performed 20 times (with different random seeds) for each matrix density. Then the average results over all the time slices (*i.e.*, 20×64 times) are reported.

We can see that our AMF approach has significant improvement over the other approaches over MRE (>41.4% for RT, >24.4% for TP) and NPRE (>65.5% for RT, >37.9% for TP), while still achieving comparable (or best) results on MAE (-0.3% $\sim$ 12.5% for RT, -7.8% $\sim$ 8.3% for TP).

TABLE I. Accuracy Comparison (A Smaller MAE, MRE or NPRE Value Means Better Accuracy)

QoS	Approach	Density = 10%			Density = 20%			Density = 30%			Density = 40%			Density = 50%		
		MAE	MRE	NPRE	MAE	MRE	NPRE	MAE	MRE	NPRE	MAE	MRE	NPRE	MAE	MRE	NPRE
RT	UPCC	0.8500	0.6484	5.4251	0.7696	0.5425	4.1452	**0.7313**	**0.5054**	3.7130	**0.7050**	**0.4801**	3.4341	0.6862	**0.4610**	3.2375
	IPCC	0.9460	0.7761	5.7514	0.8977	0.7525	5.5029	0.8573	0.7109	5.2877	0.8238	0.6807	5.0301	0.7888	0.6446	4.7026
	UIPCC	0.8482	0.6431	5.3820	**0.7719**	0.5510	4.3172	0.7332	0.5181	3.9556	0.7057	0.4944	3.6991	**0.6843**	0.4739	3.4904
	PMF	**0.8332**	**0.5283**	**2.8231**	0.7731	**0.5269**	**3.0672**	0.7443	0.5237	**3.1161**	0.7265	0.5205	**3.3160**	0.7104	0.5099	**3.0427**
	AMF	**0.7288**	**0.3096**	**0.9728**	**0.7034**	**0.2807**	**0.8994**	**0.6936**	**0.2667**	**0.8667**	**0.6892**	**0.2587**	**0.8502**	**0.6863**	**0.2542**	**0.8414**
	Imp.(%)	12.5%	41.4%	65.5%	8.9%	46.7%	70.7%	5.2%	47.2%	72.2%	2.2%	46.1%	74.4%	-0.3%	44.9%	72.3%
TP	UPCC	9.5011	1.6503	17.3322	8.4699	1.4134	16.8860	7.8835	1.2571	16.8194	7.5548	1.1595	16.8934	7.3504	1.0909	16.9664
	IPCC	9.6634	0.7859	11.4606	8.9234	0.7124	10.4361	7.9731	0.6255	8.8113	7.4345	0.5855	8.0981	7.0241	0.5556	7.6114
	UIPCC	9.3104	1.4363	15.0760	8.3855	1.2611	14.2780	7.5166	1.0947	13.2519	7.0149	1.0172	12.8740	6.6556	0.9628	12.6269
	PMF	**6.0431**	**0.4699**	**2.1754**	**5.6822**	**0.4477**	**2.4413**	**5.3076**	**0.4253**	**2.4966**	**5.0687**	**0.4012**	**2.4129**	**4.8068**	**0.3863**	**2.3976**
	AMF	**5.5427**	**0.3551**	**1.3506**	**5.4356**	**0.3178**	**1.0622**	**5.2974**	**0.3007**	**0.9607**	**5.1901**	**0.2916**	**0.9244**	**5.1809**	**0.2854**	**0.9013**
	Imp.(%)	8.3%	24.4%	37.9%	4.3%	29.0%	56.5%	0.2%	29.3%	61.5%	-2.4%	27.3%	61.7%	-7.8%	26.1%	62.4%

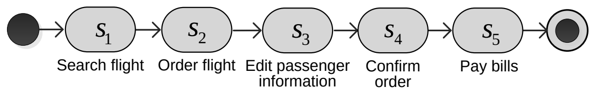


Fig. 2. A Prototype Application for Flight Purchase

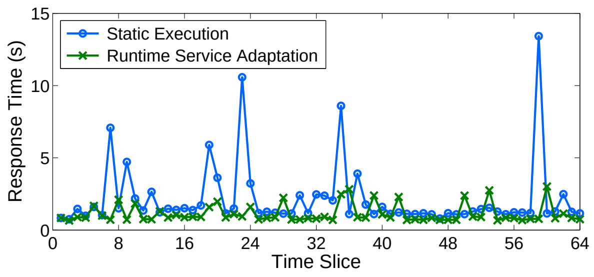


Fig. 3. The Results on Application Response Time

B. Case Study

In this section, we evaluate the performance of our online QoS prediction approach via a case study. Fig. 2 depicts a prototype application for flight ordering with a simplified workflow composed of five abstract tasks. For ease of presentation, we only consider sequential invocations in the application workflow for our case study, but our online QoS prediction approach can also be used for service adaptation of applications with other compositional structures, such as branch, loop, and parallel [5]. As shown in Fig. 2, a customer begins with flight search (s_1) by sending queries to the flight suppliers for available flight information. Then the user can order a flight (s_2) and the request is sent to the corresponding supplier, so that the supplier reserves the flight for the customer. Next the customer edits necessary passenger information (s_3), and sends an order confirmation request to the supplier (s_4). Once the order is confirmed, a payment transaction via online payment service (*e.g.*, paypal) will be launched (s_5). After the payment, the flight purchase is completed.

To evaluate the effect of our online QoS prediction approach on runtime service adaptation, we find 10 candidate Web services for each abstract task from our dataset. Then our prediction results at matrix density = 30% in Section C-A are used for runtime service adaptation decisions. Fig. 3 provides the evaluation results, which compare the application response times between static execution and runtime service adaptation. The static execution indicates the situation that the optimal service is selected for each abstract task at the first time slice, but keeps fixed during the execution during the course of all the 64 time slices. In contrast, for runtime service adaptation, after initial selection, weak services are continuously identified at each time slice and then get replaced by their candidate services with better QoS at runtime. We can find that runtime service adaptation with our online QoS prediction results can consistently maintain the overall application performance throughout the course of the application. For example, if the maximal acceptable response time is $5s$ in SLA, the application with static execution will incur SLA violations for five times. Consequently, our AMF approach for online QoS prediction can greatly facilitate the successful executions of runtime service adaptations.

References

[1] 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)*, 2014.

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

[3] H. Ma, I. King, and M. R. Lyu, "Effective missing data prediction for collaborative filtering," in *Proc. of the 30th ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2007, pp. 39–46.

[4] R. Salakhutdinov and A. Mnih, "Probabilistic matrix factorization," in *Proc. of of the 21st Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

[5] 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.*, vol. 30, no. 5, pp. 311–327, 2004.