6-2022

An Elastic Recommender Process for Cloud service recommendation scalability

Rui-dong Qi
Inner Mongolia University

Jian-tao Zhou
Inner Mongolia University

Zhuowei Wang
Guangdong University of Technology

Xiaoyu Song
Portland State University, song@ece.pdx.edu

Citation Details

Qi, R. D., Zhou, J. T., Wang, Z., & Song, X. (2022). An elastic recommender process for cloud service recommendation scalability. Concurrency and Computation: Practice and Experience, e7066.

Received: 22 September 2021 | Revised: 10 March 2022 | Accepted: 20 April 2022
DOI: 10.1002/cpe.7066

WILEY

RESEARCH ARTICLE

An elastic recommender process for cloud service recommendation scalability

Rui-dong Qi[1] | Jian-tao Zhou[1] | Zhuowei Wang[2] | Xiaoyu Song[3]

[1]College of Computer Science, Inner Mongolia University, Hohhot, Inner Mongolia, China
[2]School of Computers, Guangdong University of Technology, Guangdong, China
[3]Department of Electrical and Computer Engineering, Portland State University, Portland, Oregon, USA

Correspondence
Jian-tao Zhou, College of Computer Science, Inner Mongolia University, Hohhot, China.
Email: cszhoujiantao@qq.com

Funding information
National Natural Science Foundation of China, Grant/Award Number: 62162046; Research and Application of Key Technology of Big Data for Discipline Inspection and Supervision, Grant/Award Number: 2019GG372; The Major Project of Inner Mongolia Natural Science Foundation, Grant/Award Number: 2019ZD15

Abstract

Cloud computing services are ubiquitous in society and cloud recommender systems play a crucial role in intelligently selecting services for cloud users. Currently, recommendations are static with low scalability. Only one recommendation list is generated at a time and the recommender strategy in the recommendation cycle is not adjustable. This paper presents a new elastic recommender process (ERP) for cloud users. A Markov model is used to characterize the dynamic relationship between different user states. The ERP generates an elastic recommendation that can be used to dynamically adjust the recommender strategy to meet the user's needs based on their browsing records in the current service cycle without the recommender system's involvement. Experimental results show that the ERP improves the effectiveness of the recommender thus increasing the accuracy and diversity of its recommendations.

KEYWORDS
cloud computing, collaborative filtering, elastic recommender, service recommender, user clustering

1 | INTRODUCTION

Cloud computing provides resources in the form of services (or composite services) whose technical details are transparent to users . There are three service models: infrastructure as a service, platform as a service, and software as a service.[1] Currently, a great deal of research is being carried out on cloud services, covering aspects such as the syntax, semantics, and standards of the unified service description language,[2-4] service/application architecture,[5] service discovery and selection,[6-8] service deployment and configuration,[9,10] service composition,[11,12] service evaluation,[13,14] and so on.

Research is also being conducted on cloud service recommenders.[15] In this context, it is worth pointing out that one of the challenges faced is to choose an appropriate service (from a staggering number of cloud services) that satisfies the personalized needs of the user. A recommender system (RS) is a collection of software tools and techniques that provide users with useful suggestions.[16] Similarly, a cloud recommender system (CRS) can assess the preferences of cloud users and recommend cloud services to them (e.g., application, software, hardware, application programming interface, platform, virtual machine, database, tools).[17]

The recommendation algorithm is the key component of an RS. There are five general types of CRS algorithm, involving: collaborative filtering (CF),[18,19] content-based recommendations,[20,21] matrix decomposition,[22,23] model combination,[24] and deep learning.[25,26] Recently, integrated recommender algorithms have been found to improve the accuracy of recommendations (e.g., clustering has been integrated with CF,[27] deep learning has been integrated with content-based methods[15]).

Cloud service recommendation can be regarded as a multiobjective optimization problem.[28] The optimization goals might be, for example, accuracy, coverage, diversity, novelty and so on, each of which could be used to evaluate the performance of the recommender algorithm. Most current research tends to emphasize the accuracy of the RS. However, progressively more attention is being paid to “scalability.”

Scalability is typically investigated by experimenting with datasets of increasing size and measuring how the speed and amount of resources consumed by the RS behave as the tasks are scaled upwards.[16,29] As the number of services and need for personalized demands have increased, most traditional RS approaches have necessarily taken more time to provide their recommendations, revealing their lack of scalability. Besides, it has to be expected that such recommendations will not fulfill all the user’s demands. CF-based approaches recalculate the user’s nearest neighbors and use contextual information to provide novel recommendations. On the other hand, deep-learning-based models are designed to update the user’s features to offer recommendations. During the above processes, there is a certain amount of “inertia” associated with the recommendation provided by the RS. It thereby produces a static list that cannot adjust itself to satisfy the user’s demands.

The current problem we wish to address is how to improve the scalability of the RS and solve issues related to the static nature of the recommendations while ensuring their accuracy. This paper proposes using an elastic recommender process (ERP) to enhance the system’s scalability. The ERP can provide recommendations using shorter recommendation cycles, thus taking advantage of the user’s data history and the feedback provided by the user in real-time. Thus, the ERP provides dynamic, not static, recommendations that are adjusted using real-time contextual information collected over a short period of time in order to improve accuracy.

The rest of this paper is organized as follows. Section 2 gives a brief overview of studies related to integrated recommender algorithms. Section 3 presents and formulates the current problem of interest; the ERP is also introduced in brief. Section 4 expounds the ERP-based recommendation generation process in more detail. Section 5 outlines how the elastic adjustment process is achieved using the ERP and discusses the method’s advantages. In Section 6, experiments are carried out using the MovieLens dataset and the results used to verify our methods’ effectiveness. Finally, Section 7 sums up our work and discusses its future outlook.

2 | RELATED WORK

Many researchers have used clustering techniques to classify cloud users or services and hence generate recommendations according to each clusters’ distinctive requirements. CloudRec is a user-centered framework capable of establishing different communities for users and services.[30] The algorithm uses a multiagent paradigm, the recommender agents grouping together similar services.[31] A three-layered interactive clustering approach was proposed[32] for making personalized citation recommendations. An RS framework combining content-based and behavior-based recommenders was introduced.[33] A K-means clustering algorithm was used to aggregate past invocation data.[34] The community effect formed between users’ trusted friendships has also been used to synthesize the recommender’s features.[35]

The clustering studies mentioned above do not consider the scalability of the recommender. The K-nearest neighbor and support vector regression methods were integrated to predict the repurchases that users would make in the future.[36] That is, the temporal consumption behavior of the users was considered in order to address their future changes in behavior. A time-aware scalable collaborative model was proposed incorporating trust-clustering and scalable factorization.[37] This is important as edge services may vary over time and so a scalable temporal awareness should be considered. A scalable recommendation method (“ScaleRec”) was proposed to provide better and more rapid recommendations based on large-scale data.[38] The technique is based on using the direct and indirect trust that exists between users to partition the social big graph. Taken together, these studies suggest that the scalability issue can be solved by effectively processing the subgraphs (subclusters) using distributed technology.

A number of techniques have been developed to integrate different recommender approaches. One of the best approaches is to combine a traditional algorithm with the CF method in order to generate a recommendation. A framework for a location-aware RS (named “APPLET”) was proposed.[27] The agent paradigm was used to maintain autonomy so that interactions became more intelligent and efficient.[39] That is to say, such an approach allows a cloud service or composite service to be selected wisely. The authors propose using a user-based CF method based on Spearman coefficients to calculate the similarity between different users and thus generate recommendations.[40]

A principal component analysis–scale-invariant feature transformation algorithm was utilized to make book recommendations based on a user-uploaded image of a book or collection of books.[41] By ignoring the task similarity between different users, the authors developed a novel clustering-based and trust-aware approach to achieve personalized quality of service (QoS) predictions and reliable cloud service recommendations.[42] A concept drift-aware temporal recommendation approach was proposed to adapt to drifts in users’ preferences and provide effective recommendations.[43] The singular value decomposition method was adopted in the cloud service application program interface (API). A two-stage CF optimization method has also been proposed to improve the diversity of the recommendations made.[44]

CF-based approaches, however, require a lot of time to run because they involve numerous iterations. Similarly, their efficiency and scalability are low. The performance of an RS can be evaluated using multiple criteria. Therefore, the recommendation problem can be transformed into a multi-objective optimization problem. The result of the optimization procedure is, by definition, the best service for the user or provider in the service selection area.[45] The recommendation service can also be implemented as a multiobjective optimization problem to improve both accuracy and diversity.[28] A dynamic programming algorithm was introduced that accepts the choices of the cloud storage providers and maximizes the survival probability of the data.[46] A near-optimal approach to dynamically selecting composite services can be achieved by using an adaptive service selection method to handle the cross-cloud services.[47] A nondominated sorting genetic algorithm (NSGA), generation-based fitness evaluation strategy, and partition-based knowledge mining are used to optimize the accuracy, recall, diversity, novelty, and coverage of the recommender.[48]

A new model ("RankFromSets") was proposed that uses a stochastic optimization-based negative sampling training procedure to implement scalability.[49] The authors proposed a multiobjective optimization framework based on an NSGA to find tradeoff solutions between its two objective functions: accuracy and diversity.[50] A hybrid recommendation model was proposed that optimizes the weighting coefficients of three basic recommendation algorithms in order to enhance the robustness of the RS.[51] Although such studies have improved the efficiency of the RSs, the scalability problem has not been addressed and other criteria have also not been considered.

Another important approach to RS implementation is to utilize, either partially or entirely, a deep learning model. In order to create a scalable RS and have good coverage, the authors created a convolutional graph-embedding algorithm combined with a document-embedding model.[52] An ensemble of deep neural networks was used to generate cloud service recommendations (more specifically, the approach fuses together models based on neighborhood and neural network methods).[53] A deep neural network framework was used to create an adaptive deep latent factor model capable of adaptively learning the users' preferences.[54] A "multihead related-unit" was proposed to account for the dynamic changes that occur in viewer's and anchor's preferences on live-streaming platforms.[55] The idea is to capture and learn these properties using long short-term memory and multiple-layer perceptron models.

Artificial intelligence has also been used to optimize edge services based on data gathered via an exhaustive survey.[56] However, the training of the model is time consuming. Also, after deploying the trained model online, time is required to update the features (whether in total or in part). Hence, the current recommendation will be very similar to the previous recommendation, that is, the method suffers from inertia.

To date, many methods have been used in RS studies. The CloudRec RS adopts an adaptive QoS management regime to ensure its users experience a high level of service.[57] A brokerage-based architecture is proposed for selecting cloud services and an indexing structure, B^{cloud}, is used to manage the information relating to the service providers.[58] High performance cluster and role-based approaches have also been used for context-aware service recommendation.[59] A time-aware RS framework was established in Reference 60. A cross-domain RS has also been proposed to improve recommendation performance by partially overlapping entities.[61]

Discriminative adversarial networks are put forward to address the problem of data sparsity.[62] Diversified and scalable recommenders are proposed to improve the diversity and scalability of the service recommendation.[63] In order to help travelers choose what places to visit in a timely fashion, the authors used social media attributes to make their recommendations (URL count, emotions, followers, friends' preferences, etc.).[64] To address the growth in the number of APIs and NP-hardness of the problem of combining APIs,[65] the authors proposed an efficient privacy-preserving approach to recommending web APIs using locally sensitive hashing.

Once again, these approaches significantly improve the accuracy of the recommendation, but scalability is ignored.

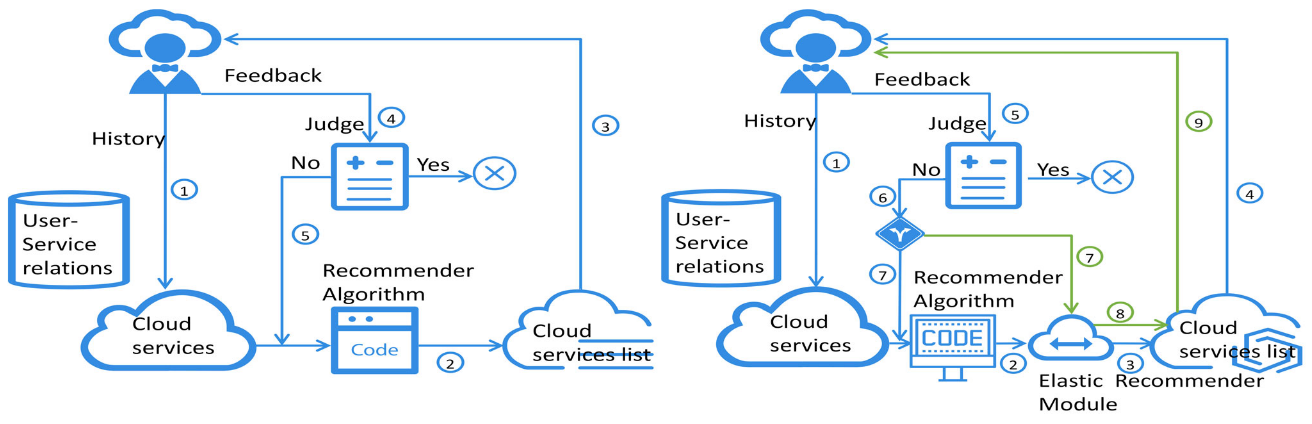
3 | PRELIMINARY INTRODUCTION TO THE ERP

In this section, we outline the problem of RS scalability, give a formal description of the method used to solve the problem, and introduce our ERP.

3.1 | Motivation and scalability problem

Assume that a user's current recommendation does not satisfy their requirements. In this case, CF-based and deep learning models need to be updated to provide a more accurate recommendation. However, it is clear that updating the model will be very time consuming as the numbers of users and services increase. Therefore, as the model is updating, the recommendations provided by the RS will suffer from inertia, that is, consecutive recommendations will be very similar and so the accuracy of the new recommendation will remain low.

To overcome this problem, recommendations should update themselves based on contextual information. In this way, the model can provide recommendations more frequently. Also, fewer computer resources will be used by the model which also improves its scalability. As an example, consider a hypothetical user called "Joy". The RS she uses recommends a service to Joy which does not meet her requirements. When the RS gets Joy's new features, it needs to calculate which users are the new nearest neighbors to Joy in order to supply recommendations. The large amount



(A) Traditional recommendation process. (B) Elastic recommendation process.

FIGURE 1 Two recommendation processes

of data that must be processed leads to the heavy consumption of computing resources and takes time. Hence, Joy has to wait some time for a new improved recommendation. During the current recommender cycle, then, Joy will continue to obtain similar recommendations that do not meet her requirements. Now assume that the recommendation can adjust itself according to contextual information. In this case, Joy will receive more suitable recommendations in a short period of time during a recommender cycle which improves the scalability of the recommendation process.

The traditional recommendation process is illustrated in Figure 1A. The RS analyzes the past experience and characteristics of the user (①) and generates a recommended service list based on its analysis (②). The recommendation is then sent to the user (③). The user then evaluates the recommendation and gives feedback (④). If the feedback is negative (i.e., the recommendation does not meet the user's requirements), it is passed back to the RS (⑤). This restarts the recommender algorithm to generate a new recommendation. When the user's feedback is received, the RS needs to update the user's features. Meanwhile, the system restarts the algorithm to provide a new recommendation. However, the RS cannot provide an accurate recommendation during this process as the user's features have not been updated. As the numbers of cloud users and cloud services have exploded, the RS has to take more time (and computer resources) to update the users' features and generate a more accurate recommendation. Therefore, the recommender cycle will become longer. That is to say, the scalability of the RS is poor.

In terms of the above example, the RS needs to analyze Joy's historical data and current contextual information in order to provide more satisfactory recommendations. However, when the amount of data is large, the time taken to process the data is long. Consequently, Joy's waiting time becomes more prolonged, her satisfaction with the service decreases, and the recommender cycle becomes longer. Moreover, scalability is mainly evaluated according to time; as the recommender cycle becomes longer and satisfaction decreases, the RS becomes less scalable.

3.2 | Elastic recommendation process

We wish to use an elastic recommendation process to overcome the problem of scalability. Figure 1B gives a brief introduction to this process.

As before, the user's service history is recorded by the cloud service (①). It is used to generate an initial recommendation and this starts the elastic recommender module (②) which generates a service list (③). This is then sent to the user (④) which generates feedback (⑤). This determines whether or not another recommendation should be provided (⑥). Negative feedback restarts the recommender algorithm and elastic recommender module (⑦). Another recommendation is then provided instantly (⑧) which is sent to the user (②). The solid green arrows in Figure 1B highlight the elastic contribution to the overall recommender process. The elastic recommender module allows more recommendations to be provided in a short period of time compared to the traditional recommendation process. The advantages of using elastic recommendations will be exhibited in the following sections.

Before we outline how elastic recommendations are generated, we first introduce some important definitions. The cloud service is taken to consist of various elementary units or essential services, as defined below.

Definition 1 (Essential services). Let s_i represent a service that cannot be decoupled. Then, the essential service set S_e is the set of all such elements, that is, $S_e = \{s_i \mid 1 \leq i \leq m\}$ where m is the total number of essential services.

As an example, a set of essential services could be: Microsoft Windows Server 2019 Base, WordPress Base Version, Ubuntu 20.04, Python 3.6, TensorFlow 1.8, CUDA 9.0, PyTorch 1.8.1, CPU, memory, storage, network, and so on. Note that different versions represent different essential services. For example, Python is a service name and 3.6 is a version number. Therefore, Python 3.6 and Python 3.8 are different essential services.

In contrast, a composite service is a collection of essential services, as defined below.

Definition 2 (Composite services). Let cs_i represent the ith composite service which consists of a collection of essential services with additional information. Thus, $cs_i = \{(s_j, h_j, w_j)|s_j \in S_e, h_j \in \mathbf{R}^+, 0 < w_j \leq 1, j \geq 1\}$, where h_j represents the quantity of s_j, $\mathbf{R}^+$ is the set of positive real numbers, and w_j is a weighting factor (such that $\sum_{j=1}^{n} w_j = 1$, $n \leq m$).

If w_j is equal to 1, then the composite service must consist of a single service. Moreover, the larger the value of w_j, the greater the effect of s_j.

As an example, consider the composite service:

cs_1 = {(TensorFlow 1.8, 1, 0.333), (Python 3.6, 1, 0.333), (CUDA 9.1, 1, 0.333)}

This composite service consists of three essential services, namely, TensorFlow 1.8, Python 3.6, and CUDA 9.1. Furthermore, $h_1 = h_2 = h_3 = 1$ and $w_1 = w_2 = w_3 = 0.333$. Another composite service could be:

cs_2 = {(TensorFlow 1.8, 1, 0.333), (Python 3.6, 1, 0.333), (CUDA 10.0, 1, 0.333)}

As CUDA 10.0 and CUDA 9.1 are different essential services, cs_1 and cs_2 are different composite services.

The composite services:

cs_3 = {(PyTorch 0.4.1, 1, 0.333), (Python 3.6, 1, 0.333), (CUDA 9.2, 1, 0.333)},

cs_4 = {(Python 3.6, 1, 0.33), (Tensorflow 2.3.1, 1, 0.333), (Pytorch 1.7, 1, 0.333)}.

are also different as they consist of different essential services.

Similarly, the service

cs_5 = {(Python 3.8, 1, 0.25), (Tensorflow 2.3.1, 1, 0.25), (Pytorch 1.8.1, 1, 0.25), (CUDA 10.2, 1, 0.25)}

is a more up to date composite service compared to cs_4 as it contains newer versions of the software components. Furthermore, if

cs_6 = {(Python 3.8, 1, 0.3), (Tensorflow 2.3.1, 1, 0.3), (Pytorch 1.8.1, 1, 0.2), (CUDA 10.2, 1, 0.2)},

then cs_5 and cs_6 are also different as their weights are different.

The next step is to define the meaning of an elastic recommendation.

Definition 3 (Elastic recommendations). Elastic recommendations are represented using five-tuples of the form: $\mathscr{E}_i = (cs_i, \mathscr{R}_i, U(*), C(*), \mathscr{A}_r)$, where cs_i is an initial recommendation, $\mathscr{R}_i$ is the relational expression between cs_i and the kth recommender strategy $\mathscr{I}_k$, $U(*)$ is a discriminant function for the preliminary service utilization (gathered from feedback), $C(*)$ is a cost function for $\mathscr{I}_k$, and $\mathscr{A}_r$ is a light-weight decision-making algorithm for making decisions in $\mathscr{I}_k$ ($0 < k \leq n, k, i \in \mathbf{N}, n = \lfloor T_i/t \rfloor, t \neq 0, t \ll T_i$, t is an elastic adjustment cycle, T_i is a recommender cycle).

The elastic recommender provides more recommendations to meet the user's requirements. In addition, $\mathscr{E}_i$ is interpreted as providing dynamic recommendations.

Suppose Joy is a computer science researcher and uses cloud services. Currently, she mainly uses Python 2.7, 3.6, TensorFlow 1.10, 1.5, 1.7, 1.8, and CUDA 9.0, 9.1. She now desires access to more recent versions of Python, TensorFlow, and CUDA and also wants to use PyTorch.

The RS will recommend cs_1 based on Joy's historical data. However, when Joy receives cs_1, she is not completely happy and will continue to search for PyTorch products. At this time, Joy's new features are being collected by the RS. During the period when Joy's features are being updated, recommendations are generated and regardless of whether the RS chooses to update their recommendations in real-time in seconds or minutes, the RS can only recommend the service cs_2 which does not contain PyTorch. If Joy wants a more desirable recommendation, then she must wait until the RS has learned and updated her features. Therefore, Joy may lose interest and select another service provider's products.

Under the same circumstances, the ERP will initially recommend cs_1 and cs_2 which will not satisfy Joy's requirements. During the period when Joy's features are being updated and recommendations generated, the ERP's three-phase process (vide infra) will adjust the recommendation strategy to provide cs_3 based on Joy's historical and real-time features. However, Joy may have accessed PyTorch 1.7 during this period and so the elastic adjustment phase will generate cs_4 in response. After updating the user's features (in seconds or minutes), the ERP will use its more complex models to generate the more accurate recommendation cs_5 which will significantly improve Joy's satisfaction with the service provider.

3.3 | Implementation of the ERP

Figure 2 presents a schematic diagram illustrating the three major phases contained within the ERP. The first phase—data preparation—contains components involved with data processing and fine-grained state processing. The user's history data is mined and analyzed to obtain a fine-grained value for the user's features. The second phase—recommendation initialization—provides an initial recommendation based on the fine-grained

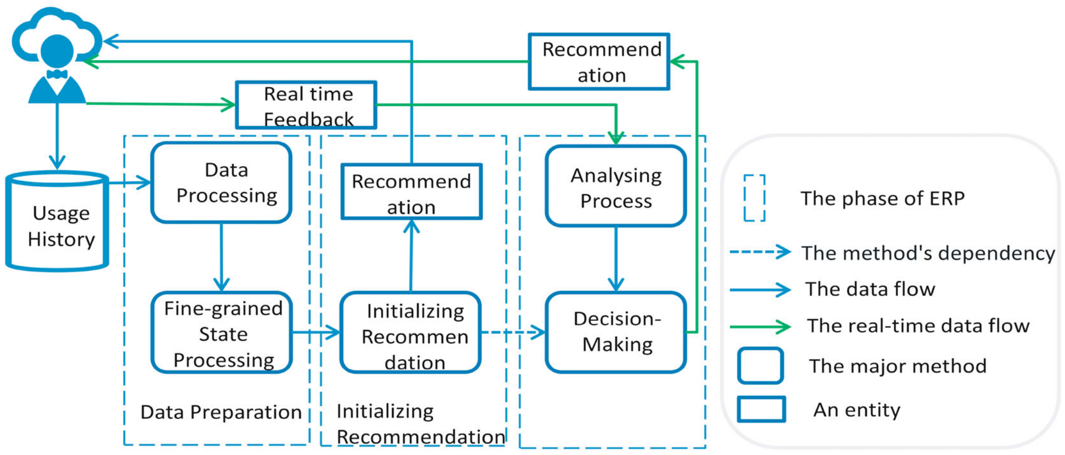


FIGURE 2 The implementation of ERP

value from the previous phase. The final phase—elastic adjustment—mainly involves analyzing the processes and decision-making. During this phase, the recommendation can be dynamically adjusted in real time according to the user's feedback.

One of the advantages of using elastic recommendations is that it improves the scalability of the RS. The elastic recommender can reconstruct the recommendation strategy (based on the user's feedback) during a much shorter recommender cycle. Therefore, the elastic recommender can proactively provide more new recommendations in a vertically elastic manner using a shorter recommender cycle. Furthermore, the elastic recommender continuously takes into account the user's feedback and fine-grained data in an optimal way in order to obtain the best trade-off between the user's requirements and recommendation.

4 | RECOMMENDATION GENERATION

This section mainly concentrates on the data preparation and recommendation initialization processes.

4.1 | Data processing

4.1.1 | User perception

In this subsection, we present the steps used to measure the service preferences of the cloud users.

As the users use the cloud services, they give feedback on their satisfaction with those services. We let $r_{i,j}$ denote the rating that user u_i gives to the *j*th composite service, cs_j. The data is first normalized using the expression

$$r^*_{i,j} = \frac{r_{i,j} - \bar{r}_i}{r_i^{\max} - r_i^{\min}}, \tag{1}$$

where $r^*_{i,j}$ is the normalized value of the rating, $\bar{r}_i$ is the mean value of the ratings from user u_i and $r_i^{\max}$ and $r_i^{\min}$ are their maximum and minimum ratings, respectively. The normalized rating values lie in the range from −1 to +1; if the value is greater than zero, then the user has given that service a positive evaluation. Conversely, if it is less than zero, then their opinion of the service is negative.

The ratings data is time-sensitive. That is, more recent ratings give a better representation of the user's current preferences. In this paper, we define a numerical parameter, a_i, to account for the impact of time on the user's ratings. This quantity is referred to as 'perceptual acceleration' and is formally defined as follows.

Definition 4 (Perceptual acceleration). The perceptual acceleration a_i can be used to give an indication of how the rating data decays with time. It is defined according to the expression

$$a_i = \int_0^{1/f(t)} \frac{1}{e^t} dt, \tag{2}$$

where $f(t_i) = \lfloor (t_0 - t_i)/t_* \rfloor + c_0$ is a decreasing function. The parameter c_0 is a constant (≥ 1) chosen to ensure that $f(t)$ is greater than 1. The other terms are time parameters: t_0 is the current time, t_i represents the rating time, and t_* is the acceleration period ($t_* \geq 1$).

As an example, suppose Joy provides ratings for services cs_i and cs_j in September 2010 and September 2020, respectively. If we assume that the acceleration period is five years, then the above two ratings are not in the same acceleration period. Therefore, the rating in September 2020 will have a higher perceptual acceleration value, that is, the more recent rating will have a higher acceleration value in the subsequent processing. The idea is that a_i can strengthen the effect of time on the ratings.

If the rating time and current time are in the same acceleration period, then this rating corresponds to the maximum perceptual acceleration value.

Proposition 1. *The maximum perceptual acceleration is given by* $\max(a_i) = 1 - e^{-1/c_0}$.

Proof. If $t_i \to t_0$, then $\lim\limits_{t_i \to t_0} \left\lfloor \frac{t_0 - t_i}{t_*} \right\rfloor = 0$ and $\lim\limits_{t_i \to t_0} f(t_i) = c_0$. Let $F(t) = a_i$, so

$$F(t) = \int_0^{1/f(t)} \frac{1}{e^t} dt, \; F(t) = -\int_0^{1/f(t)} d(e^{-t}) \text{ and so } F(t) = 1 - e^{-1/f(t)},$$

$F'(t) = e^{-1/f(t)}(f'(t)/f(t)^2)$, $f'(t) > 0$, so $F'(t) > 0$, then $F(t)$ is an increasing function, that is, the maximum of $F(t)$ is equal to $\left(1 - e^{-\frac{1}{c_0}}\right)$. ■

The maximum perceptual acceleration represents ratings that are least affected by time. Besides, if t_i and t_0 are in the same acceleration period, then the rating at time t_i has the maximum perceptual acceleration.

The time-accelerated user rating can be derived using the normalized rating value and perceptual acceleration value.

Definition 5 (Perception degree). Let $p^*_{i,j} = r^*_{i,j} \cdot a_i$ represent the perception degree of user u_i for composite service cs_j at time t_i, where $r^*_{i,j}$ is the normalized rating value, a_i is the perceptual acceleration.

As can be seen, the perception degree is the perceptual acceleration multiplied by the normalized value of the user's rating. That is, it converts the rating value into a new perceived value adjusted according to time.

In the following subsection, the rating data is first processed and then utilized to obtain the user's features.

4.1.2 | User's weighted features

One needs to deal with the issue of the nonobjectivity of the users' cloud service evaluations when deriving the weights of the user's features. Therefore, we propose the trustful perception set as given in the following definition.

Definition 6 (Trustful perceptive set). Let S^t_i represent the trustful perceptive set: $S^t_i = \{p^*_{i,j} | p^*_{i,j} \geq \overline{p}^*_i, p^*_{i,j} \geq \overline{p}^*_u, p^*_{i,j} \geq 0\}$, where $\overline{p}^*_i$ is the mean degree of perception of user u_i, and $\overline{p}^*_u$ is the mean degree of perception for all users.

The set S^t_i contains *objective* ratings and can be used to more accurately acquire the user's features.

Consider the following example. Joy has a group of perception degrees {0.8, 0.6, 0.5, 0.4, 0.2}. The mean of these values is 0.5. Any value that is higher than this mean value is considered to correspond to data that is positively correlated. We also suppose that the mean value for all users is 0.55. If the value of a perception degree is higher than the mean of all ratings, then it is assumed that nonobjectivity has been eliminated. The trustful perception set for Joy that satisfies the above conditions is {0.8, 0.6}.

Before deriving the values of a user's features, we generate the essential service set for the user which is used to record the rating data in the way defined below.

Definition 7 (Essential service set). Let $S_{i,j}$ represent the jth essential service set for user u_i, where $S_{i,j} = \left\{(h_k,\ p^*_{i,j} \cdot w_k) |\ p^*_{i,j} \in S^t_i\right\}$ where h_k and w_k belong to cs_k.

As can be seen, the jth essential service set for user u_i is made up of two-tuples of the form $(h_k, p^*_{i,j} \cdot w_k)$. For example, when computing Joy's jth dimensional feature, if Joy has used composite service cs_j containing the jth essential service (e.g., the jth essential service might be Python 3.6.) and $p^*_{i,j}$ is in S^t_i, then $(1, p^*_{i,j} \cdot 0.333)$ should be put into the set.

After the jth essential service set has been obtained, the jth weighted perception feature defined below can be formed.

Definition 8 (Weighted perceptive feature). Let the three-tuple $F_{i,j}$ represent the jth weighted perception feature of user u_i: $F_{i,j} = \left(f_{i,j}, w_{i,j}, S_{i,j}\right)$, where $f_{i,j}$ is the mean value of the jth degree of perception and $w_{i,j}$ is the weight of the jth feature.

The mean value $f_{i,j}$ is calculated using the expression

$$f_{i,j} = \frac{\sum\limits_{p^*_{i,j} \cdot w_k \in S_{i,j}} p^*_{i,j} \cdot w_k}{|S_{i,j}|}, \tag{3}$$

where $|S_{i,j}|$ is the amount of elements for $S_{i,j}$. The weight $w_{i,j}$ is given by

$$w_{i,j} = \frac{|S_{i,j}|}{\left|S^t_i\right|}, \tag{4}$$

where $\left|S^t_i\right|$ represents the size of the trustful perception set.

The *j*th weighted perception feature is used to describe the significance and perception degree of the feature. For example, suppose Joy has used composite services that do not have the *k*th essential service WordPress Base Version. So, Joy does not have the *k*th essential service and the value of the *k*th weighted perception feature is equal to 0.

Accordingly, we arrive at the following definition.

Definition 9 (Weighted features). Let the ordered m-tuple $F_i = \left(F_{i,1}, F_{i,2}, \ldots, F_{i,m}\right)$ represent the weighted features of cloud user u_i.

F_i is an m-dimensional vector, each dimension being a three-tuple $F_{i,j}$. The weighted features represent the preference data of the user. Furthermore, the weighted features are mutually independent, as expounded in Proposition 2.

Proposition 2. *Assume that the discrete random variable F_j is the jth value of u_i, where p_i is the probability of $P\left\{F_i = f_{i,i}\right\} = p_i$ $\left(0 \le p_i \le 1, \sum_{i=1}^{m} p_i = 1\right)$, then $P\left\{F_i = f_{i,i}, F_j = f_{i,j}\right\} = P\left\{F_i = f_{i,i}\right\} \cdot P\left\{F_j = f_{i,j}\right\}$ $(i \ne j)$.*

Proof. For every cs_i, where the essential services s_i and s_j are mutually independent ($s_i \in S_e, i, j \in \mathbf{N}$), i.e. for arbitrary cs_i, cs_j are mutually independent. The probability of cs_i is $P(cs_i)$. Support that cs_i contains s_k, then the conditional probability $P\{s_k|cs_i\} = P\{s_k \cdot cs_i\}/P\{cs_i\} = p_k$. There exists a set S_t, $s_k \in S_t$, and $S_t \in S^t_i$. Assuming that $P\{s_i|S_t\} = p_i$ and $P\left\{s_i|S_t,\ s_j|S_t\right\} = p_i \cdot p_j$, then $P\left\{s_i|S_t,\ s_j|S_t\right\} = P\{s_i|S_t\} \cdot P\left\{s_j|S_t\right\}$, hence, the Proposition holds. ■

4.1.3 | Clustering

To calculate the similarity between any two users (u_i and u_j, say), we use the cosine similarity function:

$$cos(F_i, F_j) = \frac{F_i \cdot F_j}{||F_i|| \cdot ||F_j||}. \tag{5}$$

Moreover, F_i is the weighted feature of u_i, $F_{i,j}$ is a three-tuple, and $f_{i,j}$ is used to calculate the similarity. Density-based clustering algorithms are currently the state-of-the-art algorithms used for clustering. In this paper, we use the "effective clustering method for finding density peaks" presented.[66] Thus, the above clustering method could obtain the user's classification, and let c_i be the *i*th clustering $(1 \le i \le n_c, n_c)$, where n_c is the total number of classification.

In this subsection, the steps have been put into place to accomplish the processing of the data. In the next subsection, we consider the fine-grained state processing step.

Remark 1. If S_i is a set and has *j* elements ($j \in \mathbf{N}$), then $|S_i| = s$, and the notation of $|S_i|$ manifests the size of S_i. $||F_i||$ presents the 2 norm of F_i. $\mathbf{N}$ is natural number. $\mathbf{R}^+$ is a set of positive real numbers.

4.2 | Fine-grained state processing

4.2.1 | Flexible state

It is not accurate to directly represent the value of a user's dimensional features with a single fixed value (as the fine details contained within the information will be lost). That is to say, some dimensions of the user's features are multivalued and so different values (within a specific range) may be needed to represent the different states of the feature. Furthermore, the value of the user feature is often a mean value, which may not reflect the users' preferences with any degree of granularity. Therefore, in this paper, we use discrete data states to describe the users' features in the elastic adjustment stage. The flexible states are defined as follows.

Definition 10 (Flexible States). Let $s_{j,k}$ be the kth flexible state of the jth weighted perception feature for user u_i. The quantity $s_{j,k}$ means the kth quantity value of the jth feature that is frequently used by the user.

The flexible state refines the features of user u_i to describe the jth weighted feature's feature. Furthermore, each flexible state has a boundary value which is given by

$$b_j = \frac{\max(h_i) - \min(h_l)}{n_s}, \; h_i, \; h_l \in S_{i,j}, \tag{6}$$

where $\max(h_i)$ is the maximum value of h_i in $S_{i,j}$ for the jth essential service, $\min(h_l)$ is the minimum value, and n_s is the number of initial states. A similar method can be used to handle the states in each cluster c_i.

The method used to derive the flexible states is outlined in Algorithm 1. As can be seen, the flexible state of the jth weighted perception feature is obtained, as well as the lower (upper) bound values, and mean of the flexible state.

In lines 1–4, the initial steps include sorting the elements, calculating boundary values, and generating sets. In lines 5–10, each element of $S_{i,j}$ is assigned to their nearest sets and $S_{i,j}$ is divided into n_s parts. In line 11, if $|S_n|/|S_{i,j}| < \rho_{\min}$, then this set should be divided. Moreover, the elements of S_n should push into the most proximal and nonempty sets. The details of the dividing process are given in lines 11–30. The point to underline here is that the 3-tuple constitutes the flexible state of this feature in line 35. The states set is generated in line 36. Furthermore, the sign of $\overline{h}$ manifests the average value, and the entire value range is $\left[(k-1) \cdot b_s + Min(h_l), k \cdot b_s + Min(h_l)\right]$. The time-cost of the solution procedure for the flexible state is $\mathcal{O}(n_s \cdot n_s)$, where n_s is the number of initial states.

As an example, suppose Joy mostly uses software and infrastructure services hosted in eastern US. Furthermore, these hosted services have memory sizes corresponding to 16, 16, 16, 16, 32, 32, 32, 32, 128, and 128 GiB. Using the Equation (6), the boundary value is 11.2, and the above list data can be assigned to three parts. Therefore, Joy has three flexible states for memory size.

This same procedure can be applied to each classification (cluster).

4.2.2 | Flexible probability

Having found the flexible states using the procedure given above, the probability of one flexible state migrating to another can be found.

Definition 11 (Flexible probability). Let $p_{k,j}$ represent the ith flexible probability that flexible state $s_{i,k}$ migrates to flexible state $s_{i,j}$ in the ith weighted perception feature of user u_i ($0 \le p_{k,j} \le 1$, $\sum_{j=0}^{\infty} p_{k,j} = 1, j \in \mathbf{N}$).

The flexible probability describes the transition probability between two flexible states; the greater the probability value, the more likely it is that transition will occur between the flexible states.

Algorithm 2 outlines how the flexible probability is calculated in practice. As can be seen, $\mathbb{P}_{i,j}$ is a symmetric matrix and so $p_{i,j} = p_{j,i}$ in lines 9–10. In lines 12–14, the distribution of the other states'flexible probabilities are derived, so the other states' probabilities are obtained. The time complexity of Algorithm 2 is $\mathcal{O}(n_s \cdot n_s)$.

The above analysis has defined the flexible states and probability. The transition sequence between flexible states has also been addressed. Moreover, the probability of the occurrence of $s_{j,k}$ does not affect $s_{j,l}$ ($l \neq k$). We therefore propose that the state transition sequence is a Markov chain.

Proposition 3. *Suppose there are n discrete random states $s_{j,k}$ $\left(1 \le k \le n, n \in \mathbf{N}\right)$ and assume that there exists a state transition sequence in a certain period of time (T_j) and that $X_k = s_{j,k}$ represents the value of the kth state sequence $s_{j,k}$ at time $P\left\{X_{k+1} = s_{j,l}\right\}$ be the probability value of X_{k+1} at time t_{k+1} and $\left\{X_k | k \in \mathbf{N}\right\}$ be a finite number of random processes. Then the sequence $\left\{X_k | k \in \mathbf{N}\right\}$ is a Markov chain.*

Proof. For two arbitrary mutually-independent flexible states $X_k = s_{j,k}$ and $X_l = s_{j,l}$ ($k \neq l$), we have $P\left\{X_{k+1} = s_{j,l} | X_k = s_{j,k}, X_{k-1} = s_{j,k}, \ldots, X_0 = s_{j,0}\right\}$ $= P\left\{X_{k+1} = s_{j,l} | X_k = s_{j,k}\right\}$. Then the probability of $X_{k+1} = s_{j,l}$ occurring only depends on $X_k = s_{j,k}$ and each $X_k = s_{j,k}$ that has occurred does not modify the probability of occurrence of the other states, that is, the sequence $\left\{X_k | k \in \mathbf{N}\right\}$ is a Markov chain. ■

T_j can be split into a number of time slices. The notation used here is that t_k represents the kth time slice ($1 < k \le l_0$). Just one state occurs in time slice t_k. The state transition sequence is a Markov chain with Markov properties. In summary, when vectorizing the users' features, the user's numerical features are represented using sequences rather than fixed values. The sequence is able to sufficiently accurately represent the actual situation of the users' features including fluctuation and change. The main idea of the elastic recommendation process is to provide recommendations using different sequence values on the same dimension and current contextual features, thus, providing recommendations in a shorter period of time. In the elastic recommendation process, the features on each dimension depend on the user's current state, i.e., each sequence of state transition is a

Algorithm 1. The solution of the flexible state

```
Input: $S_{i,j}$, $(i, j \in \mathbf{N})$.
Output: $S^{v}_{i,j}, S^{s}_{i,j}$.
1: $S = S_{i,j}$;
2: Sorting the elements of $S$;
3: Calculate $d_j$ from Equation (6);
4: Generate $n_s$ sets, and $S_n$ presents the $n$th set;
5: for each x $\in S$ do
6:     if $(n-1) \cdot b_j + Min(h_l) \leq x < n \cdot b_j + Min(h_l)$ then
7:         Push x in $S_{n-1}$, $1 \leq n \leq n_s$;
8:         Remove x in $S$;
9:     end if
10: end for
11: for $1 \leq n \leq n_s$ and $\frac{|S_n|}{|S_{i,j}|} < \rho_{\min}$ do
12:     while $S_{n-1}$ or $S_n$ exits do
13:         if $S_{n-1}$ exits and $S_n$ not exits then
14:             $S_{n-1} = S_{n-1} \cup S_n, S_n = \emptyset$;
15:         end if
16:         if $S_{n-1}$ not exits and $S_{n+1}$ exits then
17:             $S_{n+1} = S_{n+1} \cup S_n, S_n = \emptyset$;
18:         end if
19:         if $S_{n-1}$ exits and $S_{n+1}$ exits then
20:             if $x - (n-1) \cdot b_j - Min(h_l) \geq (n+1) \cdot b_j + Min(h_l) - x$ then
21:                 Push x in $S_{n-1}, x \in S_n$;
22:             else
23:                 Push x in $S_{n+1}, x \in S_n$;
24:             end if
25:             $S_n = \emptyset$;
26:         else
27:             n = n+1;
28:         end if
29:     end while
30: end for
31: for $1 \leq k \leq n_s$ do
32:     if $S_k \neq \emptyset$ then
33:         Give the lower and upper bound values;
34:         Calculate $\overline{h} = S_k$;
35:         $k = (k, \overline{h}, [(k-1) \cdot b_j + Min(h_l), k \cdot b_j] + Min(h_l))$;
36:         Set $k$ in $S^{v}_{i,j}$;
37:         Set $S_k$ in $S^{s}_{i,j}$;
38:     end if
39: end for
40: $S = \emptyset$
```

Algorithm 2. The solution of the flexible probability

```
Require: $S^s_{i,j}$.
Ensure: $\mathbb{P}_{i,j}$.
 1: $k = |S^s_{i,j}|$;
 2: for $0 \leq l < k$ do
 3:     for $0 \leq n < k$ do
 4:         if $l == n$ then
 5:             $n_l = |S_l|\ (S_l \in S^s_{i,j})$;
 6:             $n_s = |S^s_{i,j}|$;
 7:             $p_{l,n} = n_t / n_s$;
 8:         else
 9:             if $p_{n,l}! = 0$ then
10:                 $p_{l,n} = p_{n,l}$;
11:             else
12:                 $n_l = |S_l|\ (S_l \in S^s_{i,j})$;
13:                 $n_a = |\sum_{s=l}^{k} S_s|\ (S_s \in S^s_{i,j})$;
14:                 $p_{l,n} = (1 - \sum_{s=0}^{l-1} p_{s,n}) n_l / n_a$;
15:             end if
16:         end if
17:     end for
18: end for
```

Markov chain. Furthermore, the value of the current state is utilized for the recommendation. Similarly, the probability of migration between any two flexible states can be applied to each cluster (c_i).

4.3 | Initializing the recommendation process

4.3.1 | Quantitative mapping relationship

To set the amount of each essential service in the initial recommendation we use the relationship

$$h_j = \begin{cases} -\rho_r \cdot \overline{h}_u(s_{i,j}) + (1+\rho_r) \cdot \overline{h}_c(s_{i,j}), & \rho_i \leq \overline{\rho}, \\ \overline{h}_h(s_{i,j}), & \rho_i > \overline{\rho}, \end{cases} \tag{7}$$

where $\rho_r = \frac{\rho_i}{\rho_m} \log \frac{\rho_i}{\rho_m}$, ρ_i is the number of times u_i provides useful feedback, ρ_m is the maximum of all users, $\overline{\rho}$ is the mean value of all users, $\overline{h}_c(s_{i,j})$ is the mean value of u_i, and $\overline{h}_c(s_{i,j})$ is the mean value of c_i.

For example, suppose the maximum number of users feedback is 100, the mean number of feedback is 60, and Joy's is 50. Assuming the flexible state value of Joy is 0.6 and Joy's cluster is 0.4, we use Equation (7) to obtain h_j=0.46. Therefore, the amount of current essential service is 0.46 in the initial recommendation.

The mapping relationship in Equation (7) gives a quantitative connection between the users' flexible states and those of the clusters. The mapping relationship thus provides good utilization of the users' and clusters' features.

4.3.2 | Composite service generation

To generate a composite service one needs to form a combination of essential services. The generation process is based on the following principles: (i) the user's budget should be considered, (ii) the greater the weight of a certain feature, the larger the contribution it makes in the recommender.

The service cost has great significance. A cost function $C(*)$ is used to measure the cost of a recommendation:

$$C(cs_i) = \sum_{k=0}^{m} h_k \cdot s_k^c, \tag{8}$$

where $C(cs_i)$ is the entire cost of cs_i, s_k is included in cs_i, h_k is the amount of s_k and s_k^c is the cost of essential service s_k.

A group of essential services constitutes a composite service. In this work, the greedy memorial method is used to select a flexible state of a particular feature. cs_i includes s_k, then s_k should satisfy $w_k \geq \theta$ and the user's budget. Furthermore, each feature has more than one flexible state, and therefore each flexible state can be one of the solutions for the elastic recommender.

For example, suppose Joy prefers TensorFlow 1.8, Python 3.6, CUDA 9.1, PyTorch 1.8.1 in descending order. A combination of these essential services should therefore be recommended to the user. However, the first three items cost precisely the user's maximum consumption. As these items constitute cs_1, it is this composite service that should be recommended to the user.

This completes the discussion of how new recommendations are generated. We now need to show how flexible states can be used to achieve elastic adjustment.

5 | ELASTIC ADJUSTMENT PROCESS

Elastic adjustment involves two processes: analysis and decision-making.

5.1 | Analysing process

If service cs_i does not meet the user's requirements, the elastic recommender will adjust the flexible states to reconstruct the recommendation given to the user. The RS attempts to adjust the essential services by reducing the attention paid to essential services with low utilization and increasing that paid to essential services that are important to the user.

The preliminary utilization ratio of each essential service is calculated using the expression

$$r_i = \frac{h_i^*}{h_i}, \tag{9}$$

where h_i comes from the ith essential service of cs_i and h_i^* is the amount of the ith essential service in the preliminary demand, given by

$$h_i^* = \sum_{s_i \in cs_j, cs_j \in L_o} h_i \cdot \frac{cos(cs_i)(cs_j)}{\sum_{cs_j \in L_o} cos(cs_i)(cs_j)}, \tag{10}$$

where $cos(cs_i)(cs_j)$ is the cosine similarity between cs_i and cs_j. L_o is a list of services that the user has clicked ($L_o \in L$, where L is the recommendation list).

If $r_i = 0$, then the ith essential service is not used, so $S_{r=0} = \{s_i | r_i = 0\}$ is the set of all such services. If $0 < r_i \leq 1$, then the greater the value of r_i, the greater the amount of the ith essential service in the composite service. The set $S_{r\leq\alpha} = \{s_i | r_i \leq \alpha\}$ is the set of services with utilization ratios less than α.

However, if $r_i \geq 1$, the amount of ith cloud service available in the composite service is insufficient. If $h_i = 0$ then the recommendation does not contain the ith essential service but it is necessary. The set $S_{s=0} = \left\{s_i | \{s_i | \{s_i \in cs_j\} \cup \{s_i \in cs_k\}, cs_j, cs_k \in L_o\} / \{s_i | s_i \in cs_i\}\right\}$ represents the set of all such services. Meanwhile, the set $S_{r>1} = \{s_i | h_p > 0\}$ is the set of services in which s_i is deficient.

For example, suppose we have recommended a service list to Joy in which the first two services in the recommended list, TensorFlow 1.8 and Python 3.6, are essential services. (Recall that the services at the top of the recommendation list have higher relevance for the user concerned.) Further suppose that the second half of the services on the recommendation list include PyTorch 1.8.1 and Python 3.8, and that PyTorch 1.8.1 is an essential service that Joy has clicked on. Then, the RS should recommend both PyTorch 1.8.1 and Python 3.8 to Joy. Therefore, the elastic adjustment process should add the service PyTorch 1.8.1 to the top of the recommendation list.

The following subsection is used to decide which essential services need to be adjusted.

5.2 | Decision-making process

The decision-making process also has two processes. The preliminary decision process chooses the flexible states based on the preliminary utilization of the essential services:

$$N(d_i) = \begin{cases} \left[(1-\beta)p_{k,j} - \beta d_i\right] nd_i, & d_i < 0, \\ \left[(1-\beta)p_{k,j} + \beta d_i\right] ne_i, & d_i > 0, \end{cases} \tag{11}$$

where $N(d_i)$ is also a serial number, $d_i = 1 - r_i$, nd_i, nd_i is the number of decreasing flexible states, ne_i is the number of increasing flexible states, $p_{k,j}$ is the probability of the flexible state changing from k to j, and the value of $p_{k,j}$ in $\mathbb{P}_{i,j}$.

Essential services that have low preliminary utilization levels (or have never been used at all) should be reduced and vice versa.

The above process gives a preliminarily set of flexible states that could be alternated between. The next step is to decide which initial elastic states should be increased and which should be decreased. The formal process for deciding which flexible states to choose is expressed in the form of a minimization problem:

The above process gives flexible states that could be alternated preliminarily. Which initial elastic decision states can be increased or decreased? Therefore, the final decision process of choosing the flexible state is formalized as a minimization problem in Equation (12).

$$\begin{aligned} &\min = \left\{ \max \left[\sum_{s_j}^{S_{r=0}} C(s_{j,k} \to s_{j,t}) + \sum_{s_j}^{S_{r\le\alpha}} C(s_{j,k} \to s_{j,t}) \right] - \left[\sum_{s_j}^{S_{s=0}} C(s_{j,k}) + \sum_{s_j}^{S_{r>1}} C(s_{j,k} \to s_{j,t}) \right] \right\}, \\ &s.t. \quad C(cs_i) \le C_B, \quad U(\mathscr{I}_k) > U(\mathscr{I}_{k+1}), \end{aligned} \tag{12}$$

In Equation (12), $C(s_{j,k} \to s_{j,t}) = C(s_{j,k})$ - $C(s_{j,t})$, so that $C(s_{j,k} \to s_{j,t})$ represents the cost change on moving from one flexible state, $s_{j,k}$, to another, $s_{j,k}$. The flexible state $s_{j,k}$ is transformed to $s_{j,t}$ using Equation (11). Equation (12) minimizes the difference between the maximum cost of transforming the low-level flexible states and the increased cost of the new flexible states. The second part contains the constraints on the minimization problem.

The symbol C_B in Equation (12) represents the user's budget. The discriminant function $U(cs_i)$ evaluates the mean waste rate in the composite service or elastic recommender strategy:

$$U(cs_i) = \frac{\sum_{i=1}^{w} |r_i - 1|}{w}, \tag{13}$$

$U(cs_i)$ is used to calculate each elastic recommender strategy of $\mathscr{E}_i$.

As already stated, the decision-making process is transformed into a minimization problem. Algorithm 3 presents the terminal decision algorithm determining which flexible states should transform. It is based on the application of the memorial greedy algorithm to select the cloud services that are in urgent need of improvement. In lines 3–10, the total of services can release. In lines 11–18, the total of cloud services should improve. Lines 19–29 guarantee that at least one flexible state can make a transition. The priority of the services should be considered, and so, in line 30, the services are sorted according to the user's service requirements. Line 32 presents the constraint conditions. In lines 31–40, the kth elastic recommendation is generated. As w is a constant less than m, the time complexity of Algorithm 3 is $\mathcal{O}(w^2)$.

Figure 3 shows an example illustrating how migration occurs between flexible states. As can be seen, the jth feature has three flexible states: $s_{j,1}$, $s_{j,2}$, and $s_{j,3}$. In Figure 3A, the blue circle indicates that $s_{j,1}$ makes up the composite service. In Figure 3B, the blue broken circle around $s_{j,1}$ indicates that it has a low selection rate. Therefore, $s_{j,1}$ should be adjusted. There are two alternative flexible states as indicated by the red dotted arrows. Equation (11) is therefore used to select an alternate primary flexible state. A solid red arrow is used to point to a flexible state that can be migrated to (Figure 3C). The above steps choose the flexible states that can migrate.

A decision-making process will also be undertaken to determine which flexible states of the different essential services can migrate. In Figure 3D, assuming that the flexible state of the jth essential service can migrate, the continuous blue circle moves to represent the migration of the flexible state from the first state to the second. It is worth reiterating at this point that the migration process outlined above is based on the fine-grained states of each essential service.

5.3 | Elastic recommendations

Based on the above statements, this subsection puts forward an important theorem related to the elastic recommendation process.

The elastic recommendation $\mathscr{E}_i$ contains five elements. Therefore, it is not just the result of one recommendation and can generate recommendations dynamically. The composite service cs_i is an initial recommendation. The preliminary utilization of each essential service can be calculated

Algorithm 3. The elastic adjustment process

```
Require: $cs_i, C(*),\ r_j(1 \leq j \leq w)$.
Ensure: $\mathscr{I}_{k+1}$.
 1: $d_t = 0, d_e = 0, R_d = \{\ \}, R_e\{\ \}$;
 2: $t_s = 0, t_c = 0$;
 3: for $s_t \in S_{r=0}$ do
 4:     $d_t = d_t + C(s_{j,k} \rightarrow s_{j,t})$;
 5:     $R_d.append([j][1 \cdot w_j])$;
 6: end for
 7: for $s_t \in S_{r<\alpha}$ do
 8:     $d_t = d_t + C(s_{j,k} \rightarrow s_{j,t})$;
 9:     $R_d.append([j][(1 - r_j) \cdot w_j])$;
10: end for
11: for $s_t \in S_{r_j>1}$ do
12:     $d_e = d_e + C(s_{j,k} \rightarrow s_{j,t})$;
13:     $R_e.append([j][(r_j - 1) \cdot w_j])$;
14: end for
15: for $s_t \in S_{s=0}$ do
16:     $d_e = d_e + C(s_{j,t})$;
17:     $R_e.append([j][1 \cdot w_j])$ ;
18: end for
19: while $d_t = 0$ or $d_t < min(R_e[][1])$ do
20:     for $1 \leq j \leq w$ do
21:         if $r_j < (\alpha + \triangle\alpha)$ then
22:             $d_t = d_t + C(s_{j,k} \rightarrow s_{j,t})$;
23:             $R_e.append([j][(1 - r_j) \cdot w_j])$;
24:         end if
25:         if $d_t == d_t + \Delta d_t$ then
26:             break;
27:         end if
28:     end for
29: end while
30: $R_d = sort.R_d([\ ][1]), R_e = sort.R_e([\ ][1])$ ;
31: for $t \in R_e$ do
32:     if $t[1] <= d_t - t_s$ and $t_c + C(s_{j,k} \rightarrow s_{j,t}) < C_B$ and $U(k) > U(k + 1)$ then
33:         $t_s = t_s + t[1]$;
34:         $t_c = t_c + C(s_{j,k} \rightarrow s_{j,t})$;
35:         FnMU($t[0]$);
36:         Push $s_t$ in $\mathscr{I}_{k+1}$;
37:     else
38:         break;
39:     end if
40: end for
```

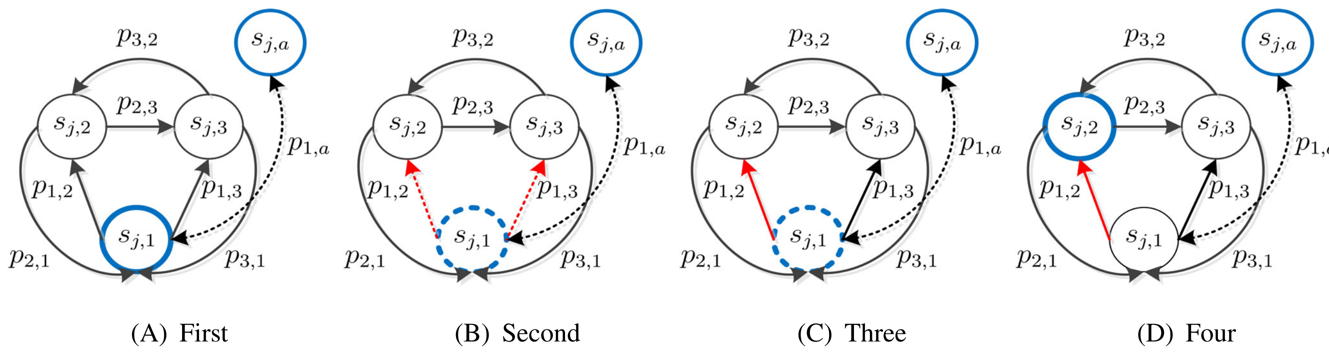


FIGURE 3 Preselected interval for the service

using Equation (9). In Equation (11), the preliminary decision process is utilized to choose flexible states. Algorithm 3 is a lightweight decision-making algorithm for the migration of the flexible states. The following discusses some of the fundamental properties of the elastic recommendations.

The discriminant function is used to calculate the waste rate of the recommender strategy in $\mathscr{E}_i$. Lemma 1 gives the relationship between the waste rates in adjacent recommender strategies.

Lemma 1. *$\mathscr{I}_k$ is rebuilt by $\mathscr{E}_i$ at the kth time slice based on cs_i $(1 \leq k \leq \lfloor T_j/t \rfloor)$ and the discriminant function $U(\mathscr{I}_k)$ is aperiodic.*

Proof. When the initial service cs_i is generated, and if $r_i = \alpha(1 \leq i \leq m)$ corresponds to low usage or inadequate service, then $\mathscr{E}_i$ generates $\mathscr{I}_1$ at t_1 provided $U(cs_i) > U(\mathscr{I}_1)$. $\mathscr{I}_k$ is generated and $U(\mathscr{I}_k) > U(\mathscr{I}_s)$. If $k < s$, then $U(\mathscr{I}_k) > U(\mathscr{I}_s)$ and each $U(\mathscr{I}_k)$ is different, hence, $U(\mathscr{I}_k)$ is aperiodic. Accordingly, the conclusion holds. ■

Lemma 1 declares that each $U(cs_i)$ is different and $U(cs_i) > U(\mathscr{I}_1)$. For each (cs_i) and elastic recommendation $\mathscr{I}_k$, $U(*)$ is a decreasing function. Provided that $k < s$, then $U(\mathscr{I}_k) > U(\mathscr{I}_s)$ and $U(cs_i) > U(\mathscr{I}_k)$. If $\mathscr{I}_{k+1}$ has occurred, then $\mathscr{I}_k$ must have at least one flexible state that has transitioned.

Theorem 1 expounds the other property—the bounded nature of the elastic strategies.

Theorem 1. *Suppose $\mathscr{I}_i^k$ has occurred $(k \geq 1)$, then $U(\mathscr{I}_i^k)$ must have the lower bound.*

Proof. Suppose the user has a fixed budget C_B. The strategy $\mathscr{I}_k$ transforms cs_i and has the properties given in Lemma 1. Supposing $k \to +\infty$, $C_B \to +\infty$ and $U(cs_i) = \alpha$ $(0 \leq \alpha \leq 1)$. Then, $U(\mathscr{I}_1) > \alpha - \triangle\alpha$ and $U(\mathscr{I}_k) = \alpha - k\cdot \geq \triangle\alpha$ $(0 \leq \alpha - k \cdot \triangle\alpha < 1)$. If $0 \leq \alpha - k \cdot \triangle\alpha$ and $\lim_{k\to\infty} \frac{\alpha}{k} = 0$, then $\triangle\alpha = 0$. Hence, the assumption is incorrect. Furthermore, if T_i is finite and decreases monotonically, then there is a finite minimum value of $U(\mathscr{I}_k)$. ■

Theorem 1 expounds that $U(\mathscr{I}_k)$ has a lower bound. Meanwhile, it declares that the number of elastic strategies is bounded.

The above two sections introduce the details of the ideas and methods used to carry out the ERP. Note that, the elastic RS generates a new $\mathscr{E}_i$ for the user. Thereafter, $\mathscr{E}_i$ can use its five elements to generate a new recommender strategy without restarting the RS. Thus, the principal purpose of the ERP is to improve the scalability of the recommendation system.

6 | PERFORMANCE EVALUATIONS

6.1 | Experimental details

Dataset: The proposed method was evaluated by performing experiments using the "ml-latest-small" dataset which is an extensive real-world dataset produced by MovieLens since October 2016.[67] This dataset contains information relating to 610 users, 9742 movies, and 100,836 user ratings. Each user has rated at least 20 movies, with ratings ranging from 1 to 5. A method of cross-validation was used to subdivide the dataset into ten mutually-disjoint portions. Of these, eight portions were selected to use as the training set; the other two formed the test set.

Comparative methods: Our proposed ERP method is compared to six baseline algorithms to investigate its performance:

- The random strategy (TRS)—one of the most straightforward and rapid recommendation strategies. A new recommendation is randomly selected from the existing services.

- The most-highly rated strategy (TMRS)—identifies representative users and items based on a matrix factorization method to complete recommendations. It achieves a good balance between coverage and diversity.[68]
- Personalized cloud service recommendation (P-CSREC)—uses a similarity measure based on multiple meta-paths and FP-growth association rules for personalized service recommendations.[69]
- Fast recommendation with M-distance (FRMD)—a fast recommendation method based on the M-distance between the average ratings of two items.[70]
- Diversity balancing for two-stage collaborative filtering (DBTS)—uses a two-stage optimization strategy that uses social relationships with CF to balance accuracy and diversity.[44]
- Knowledge graph recommendation method (KG2Rec)—uses knowledge graphs to obtain vectors for the users and locally sensitive hashing techniques to rapidly calculate the similarity between them.[71]

The TRS, TMRS, and FRMD approaches provide fast recommendations. P-CSREC is a cluster-based method and DBTS features improved recommendation diversity. TMRS, P-CSREC, and DBTS are CF-based methods. KG2Rec integrates machine learning and also reduces the recommendation time.

Parameter setting: In the experimental dataset, there are 19 different gene categories (e.g., action, adventure, animation). The number of essential services equals the number of gene categories, so $m = 19$.

Suppose the initial utilization rate of an essential service exceeds α. In this case, this essential service can satisfy the user's demands and so it does not need to be adjusted. Therefore, if the value of α is set too high, there will be very few adjustments made. However, if α is low, there will be too many potential solutions generated. Hence, a compromise is made and α =0.75 is chosen.

The elastic adjustment process mainly relies on the user's historical experience. Overall, the historical data accounts for 60% of the recommendation process (so the current contextual information accounts for 40%). By choosing $\beta = 0.6$, we can make sure the proportion is 40% for the history action.

The weights of the essential services of the users are calculated during the initial recommendation process. A small weight means that the user's use of that service is negligible. Hence, essential services with weights less than $\theta = 0.2$ are not selected.

The user rating is divided into five categories. The initial number of resilient states is taken to be twice the number of rating categories, so n_s=10, including the above categories and initial refinement. When classifying all the data, the initialization process ensures that each class contains at least 10% of the data (i.e., $\rho_{\min} = 0.1$), ensuring that there are ten classes at most.

Implementation: Python software was used to implement our proposed scheme. More specifically, the IntelliJ IDEA Community Edition v2018.1.1 x64 and Python v3.6.5 packages are used. A computer equipped with two Intel Xeon-X5760s central processing units (2.93 GHz) and 64 GB of memory was employed. The operating system used on the computer was Windows 2008R2 Server Enterprise 64.

6.2 | Experiments

Three experimental aspects are used to compare and evaluate our proposed approach: prediction accuracy, average recommendation time, and recommendation diversity.

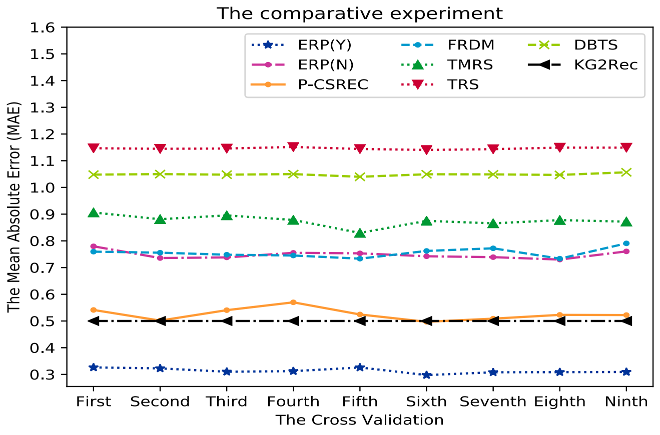
6.2.1 | Prediction accuracy

The accuracy of the predictions can be assessed using two quantitative measures: the mean absolute error (MAE) and root mean square error (RMSE):[72]

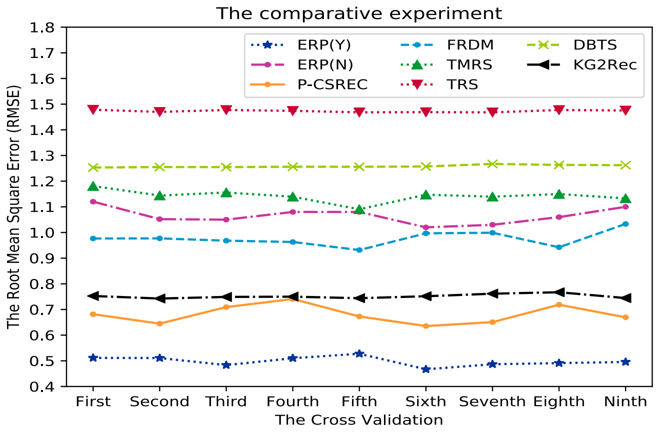
$$MAE = \sum_{i \in N} \frac{|cs_i - \hat{cs}_i|}{|N|}, \tag{14}$$

$$RMSE = \sqrt{\sum_{i \in N} \frac{(cs_i - \hat{cs}_i)^2}{|N|}}, \tag{15}$$

where cs_i corresponds to the cloud user's real value and $\hat{cs}_i$ to the predicted value . Clearly, lower MAE and RMSE values imply a better (more accurate) recommendation.



(A) Comparison of the results (MAE values).



(B) Comparison of the results (RMSE values).

FIGURE 4 The MAE and RMSE

The results of the comparison experiments, from the point of view of accuracy, are given in Figure 4A,B. In these diagrams, the results are presented along the abscissae according to the way the dataset is partitioned (for instance, the results labeled "first" correspond to the first way the data was partitioned).

Note that the ERP results are presented in two forms labeled ERP(N) and ERP(Y). The former results (blue dotted lines) correspond to those obtained when the elastic adjustment process is not used; the latter (purple dash–dot lines) to when it is. This separation makes it clear that using the elastic adjustment process can significantly improve the accuracy of the recommender. Furthermore, the ERP(Y) MAE and RMSE values are lower than those achieved using the other algorithms (by 20%).

According to the MAE and RMSE results, our model has improved accuracy compared to the other methods. There are four reasons for this: (i) the trustful perception set filters out untrustworthy data; (ii) the perception acceleration process allocates more weight to more recent data; (iii) by using clusters, users in the same cluster enhance the generation of good recommendations; (iv) the ERP employs flexible states that can provide more recommendations in a short period of time.

6.2.2 | Time costs

Time cost is used to measure the efficiency and scalability of the recommendation process. The lower the time cost, the more efficient the recommendation process and the better its scalability.

The time taken to load all the data into memory and complete the last recommendation was statistically analyzed. The mean recommendation time t_a^e is calculated using

$$t_a^e = \frac{T_a}{|N|}, \tag{16}$$

where T_a is the time cost of the recommendation process and N is the number of recommendations.

Table 1 presents the mean recommender time costs derived using the experimental times recorded for the seven algorithms (values in milliseconds). In Table 1, the first column labeled ERP(N) effectively gives the average time to generate the initial recommendation and the second column labeled ERP(Y) gives the average time when elastic recommendations are made. Considering just the first row of data, it is clear that the time taken to generate an initial elastic recommendation, 562.5 ms, is much longer than the time cost of each elastic recommendation, 2.226 ms. Besides this, the average recommendation time is the recommendation cycle time of each algorithm.

Table 1 indicates that the recommendation time of the elastic recommender algorithm is shorter than those of the other algorithms. Therefore, when the services provided to a user do not to satisfy their requirements, the elastic recommendation process has enough time to generate a new recommendation. If the ERP does not meet the user's requirements, then it can generate a new strategy $\mathscr{I}_i^k$ in a short period of time. Our statistical analysis reveals that the time cost for the elastic recommendation obtained using our ERP algorithm is less than a traditional recommendation cycle. Hence, the elastic recommender is a feasible basis for a practical RS.

In summary, our statistical results reveal that the time cost for the ERP is less than the recommender cycles of foregoing processes. The flexible states employed permit recommendations to be generated quickly. The ERP generates different recommender strategies during a short recommender cycle in order to improve the performance of the recommender.

TABLE 1 Average recommendation times

	Algorithm (millisecond)							
Cross	**ERP(N)**	**ERP(Y)**	**P-CSREC**	**FRMD**	**TRS**	**TMRS**	**DBTS**	**KG2Rec**
First	562.5	1.126	619.5	727.1	1.4	6.658	930.1	305.9
Second	559.0	0.98	480.7	727.6	1.423	6.639	935.9	318.5
Third	555.7	1.049	453.9	727.9	1.44	5.976	945.9	314.9
Fourth	567.0	0.943	438.1	721.1	1.432	5.003	950.2	331.1
Fifth	561.4	0.973	547.6	718.8	1.422	5.000	933.5	311.8
Sixth	567.9	0.995	582.3	723.0	1.431	5.164	936.0	311.4
Seventh	566.2	0.982	498.2	724.4	1.483	4.887	933.2	316.2
Eighth	563.8	0.995	583.1	716.4	1.448	4.959	928.6	316.9
Ninth	569.5	1.052	556.6	714.9	1.421	4.948	917.6	310.2

6.2.3 | Diversity

Experiments were performed to measure the diversity of the essential services recommended. The diversity of a recommendation is measured using the formula

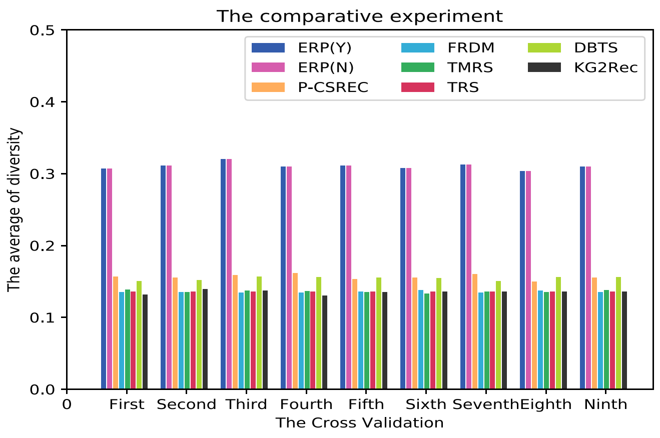
$$D_i = \frac{|s_i^o|}{|s|}, \tag{17}$$

where s is the set of essential services and s_i^o is the number of essential services in the ith elastic recommendation. The larger the value of D_i, the greater the diversity of the recommendation.

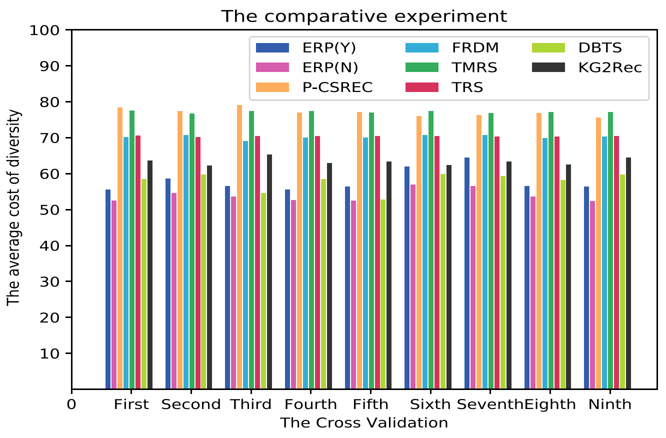
Figure 5A shows the average diversity values calculated for the different recommender algorithms. It is clear that the highest diversity rates (blue and purple columns) are produced using our ERP algorithm. Moreover, regardless of whether the ERP uses the elastic adjustment process or not, the ERP produces results with much higher degrees of diversity than the other algorithms.

It is important to determine if this improved diversity performance comes at a price. Hence, the cost of each recommendation is compared with its diversity value to determine a measure of the average cost per unit diversity, C_d:

$$C_d = \frac{1}{|L|} \sum_{cs_i \in L} C(cs_i) \cdot D_i^{-1}, \tag{18}$$



(A) Average diversity values.



(B) Average cost per unit diversity for the different algorithms.

FIGURE 5 The average of diversity and average cost of diversity

where L is the recommendation list. Clearly, the lower the value of C_d, the better the efficiency of the RS.

Figure 5B shows the average C_d values calculated for the different algorithms. As can be seen, ERP(Y) and ERP(N) produce the lowest values among the various methods. This result is a bit counterintuitive. On the one hand, the elastic recommender process can shut down or reduce the magnitude of essential services, making them unused or underutilized. On the other hand, the use of flexible states gives a much more accurate description of the nature of the cloud users. Moreover, ERP chooses services for a purpose, and so it has a lower C_d value than the other algorithms.

In general, these results indicate that ERP has a higher accuracy but a lower cost of diversity than the other algorithms. Therefore, the elastic recommender is feasible. To sum up, the elastic recommender process is a solution of elasticity recommendation for service recommendation.

7 | CONCLUSIONS

This paper proposes an elastic service recommender process, called ERP, that is capable of making accurate service recommendations with improved scalability. The ERP can construct many recommender strategies in a short period of time in order to improve scalability and generate recommendations in real-time. As a result, there is less inertia in the feature update process.

The elastic recommendations generated are more dynamic and diverse than more conventional approaches which simply generate static recommendation lists when they update their features. Fine-grained features are incorporated into the elastic recommendation reconstruction process. This allows the recommender to offer a broader range of strategies which ensures that the ERP can obtain the best tradeoff between accuracy and diversity. The experimental results show that the ERP reconstructs the recommendation process which makes it much faster (by three orders of magnitude) than baseline algorithms of similar accuracy. Moreover, the average diversity is improved by almost 50%, that is, there is much less inertia in the recommendation process. Finally, the ERP method is much more accurate than the next most accurate baseline algorithm (MAE and RMSE values improved by 18% and 10%, respectively).

The test data used in this study comes from the MovieLens database. This may, of course, not accurately reflect the actual cloud usage data of specific groups of users or customers. To broaden the range of applicability of the new ERP approach, further work is required to establish additional service platforms and collect more data for analysis. Moreover, reinforcement learning can be utilized to obtain relationships between flexible states. The "cold start" problem inherent in the elastic recommender process can then be addressed.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China (62162046), the Major Project of Inner Mongolia Natural Science Foundation (2019ZD15), Research and Application of Key Technology of Big Data for Discipline Inspection and Supervision (No.2019GG372). It is also sponsored in part by National & Local Joint Engineering Research Center of Intelligent Information Processing Technology for Mongolian, Engineering Research Center of Ecological Big Data, Ministry of Education, Inner Mongolia Engineering Laboratory for Cloud Computing and Service Software, Inner Mongolia Key Laboratory of Social Computing and Data Processing, Inner Mongolia Engineering Laboratory for Big Data Analysis Technology.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in [MovieLens Datasets] at https://doi.org/10.1145/2827872, reference number [67].

ORCID

Rui-dong Qi https://orcid.org/0000-0001-6764-8205

Zhuowei Wang https://orcid.org/0000-0001-6479-5154

REFERENCES

1. Mell PM, Grance T. SP 800-145. The NIST definition of cloud computing; 2011.
2. Nawaz F, Mohsin A, Janjua NK. Service description languages in cloud computing: state-of-the-art and research issues. *Serv Orient Comput Appl*. 2019;13(2):109-125.
3. Sun L, Ma J, Wang H, Zhang Y, Yong J. Cloud service description model: an extension of USDL for cloud services. *IEEE Trans Serv Comput*. 2018;11(2):354-368.
4. Al-Sayed MM, Hassan HA, Omara FA. CloudFNF: an ontology structure for functional and non-functional features of cloud services. *J Parallel Distrib Comput*. 2020;141:143-173.
5. Kratzke N. A brief history of cloud application architectures. *Appl Sci*. 2018;8(8):1368.
6. Parhi M, Pattanayak BK, Patra MR. A multi-agent-based framework for cloud service discovery and selection using ontology. *Serv Oriented Comput Appl*. 2018;12(2):137-154.

7. Al-Sayed MM, Hassan HA, Omara FA. An intelligent cloud service discovery framework. *Futur Gener Comput Syst*. 2020;106:438-466.
8. Nabli H, Djemaa RB, Amor IAB. Efficient cloud service discovery approach based on LDA topic modeling. *J Syst Softw*. 2018;146:233-248.
9. Toosi AN, Sinnott RO, Buyya R. Resource provisioning for data-intensive applications with deadline constraints on hybrid clouds using Aneka. *Futur Gener Comput Syst*. 2018;79:765-775.
10. Liang H, Du Y, Gao E, Sun J. Cost-driven scheduling of service processes in hybrid cloud with VM deployment and interval-based charging. *Futur Gener Comput Syst*. 2020;107:351-367.
11. Shi T, Ma H, Chen G, Hartmann S. Location-aware and budget-constrained service deployment for composite applications in multi-cloud environment. *IEEE Trans Parall Distrib Syst*. 2020;31(8):1954-1969.
12. Gavvala SK, Jatoth C, Gangadharan G, Buyya R. QoS-aware cloud service composition using eagle strategy. *Futur Gener Comput Syst*. 2019;90:273-290.
13. Zhou P, Meng LM, Qiu XS, Wang ZS, Wang ZP, Chen ZF. Evaluation of cloud service reliability based on classified statistics and hierarchy variable weight. *J Signal Process Syst*. 2019;91(10):1115-1126.
14. Kumar RR, Mishra S, Kumar C. A novel framework for cloud service evaluation and selection using hybrid MCDM methods. *Arab J Sci Eng*. 2018;43(12):7015-7030.
15. Shi M, Tang Y, Liu J. Functional and contextual attention-based LSTM for service recommendation in mashup creation. *IEEE Trans Parallel Distrib Syst*. 2019;30(5):1077-1090.
16. Ricci F, Rokach L, Shapira B, Kantor PB. *Recommender Systems Handbook*. Berlin, HD: Spring-Verlag. 1st ed. 2010.
17. Aznoli F, Navimipour NJ. Cloud services recommendation: Reviewing the recent advances and suggesting the future research directions. *J Netw Comput Appl*. 2017;77:73-86.
18. Botangen KA, Yu J, Sheng QZ, Han Y, Yongchareon S. Geographic-aware collaborative filtering for web service recommendation. *Expert Syst Appl*. 2020;151:113347.
19. Jeunen O, Verstrepen K, Goethals B. Efficient similarity computation for collaborative filtering in dynamic environments. Proceedings of the 13th ACM Conference on Recommender Systems; 2019:251-259.
20. Ullah F, Zhang B, Khan RU. Image-based service recommendation system: a JPEG-coefficient RFs approach. *IEEE Access*. 2020;8:3308-3318.
21. Lops P, Jannach D, Musto C, Bogers T, Koolen M. Trends in content-based recommendation : preface to the special issue on recommender systems based on rich item descriptions. *User Model User-Adap Inter*. 2019;29(2):239-249.
22. Samanta P, Liu X. Recommending services for new mashups through service factors and top-k neighbors. Proceedings of the IEEE. 2017 IEEE International Conference on Web Services (ICWS); 2017:381-388.
23. Zhu J, He P, Zheng Z, Lyu MR. Online QoS prediction for runtime service adaptation via adaptive matrix factorization. *IEEE Trans Parallel Distrib Syst*. 2017;28(10):2911-2924.
24. Cano E, Morisio M. Hybrid recommender systems: a systematic literature review. *Intell Data Anal*. 2017;21(6):1487-1524.
25. Haytamy SS, Omara FA. A deep learning based framework for optimizing cloud consumer QoS-based service composition. *Computing*. 2020;102(5):1117-1137.
26. Zhang S, Yao L, Sun A, Tay Y. Deep learning based recommender system: a survey and new perspectives. *ACM Comput Surv*. 2019;52(1):5.
27. Xindi MA, Hui LI, Jianfeng MA, et al. APPLET: a privacy-preserving framework for location-aware recommender system. *Sci China*. 2017;60(9):092101.
28. Shuai D, Xia C, Wang C, Wu D, Zhang Y. Multi-objective optimization based ranking prediction for cloud service recommendation. *Decis Support Syst*. 2017;101:S0167923617301173.
29. Liang Y, Lan Y. TCLBM: A task chain-based load balancing algorithm for microservices. *Tsinghua Sci Technol*. 2021;26(3):251-258. doi:10.26599/TST.2019.9010032
30. Qi Y. CloudRec: a framework for personalized service recommendation in the cloud. *Knowl Inf Syst*. 2014;43(2):1-27.
31. Mahmood A, Shoaib U, Sarfraz MS. A recommendation system for cloud services selection based on intelligent agents. *Ind J Sci Technol*. 2018;11(9):1-6.
32. Cai X, Han J, Li W, Zhang R, Pan S, Yang L. A three-layered mutually reinforced model for personalized citation recommendation. *IEEE Trans Neural Netw Learn Syst*. 2018;29(12):6026-6037.
33. Djiroun R, Guesssoum MA, Boukhalfa K, Benkhelifa E. A novel cloud services recommendation system based on automatic learning techniques. Proceedings of the 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA); 2018:598-605.
34. Silic M, Delac G, Srbljic S. Prediction of atomic web services reliability for QoS-aware recommendation. *IEEE Trans Serv Comput*. 2015;8(3):425-438.
35. Deng S, Huang L, Xu G, Wu X, Wu Z. On deep learning for trust-aware recommendations in social networks. *IEEE Trans Neural Netw Learn Syst*. 2017;28(5):1164-1177. doi:10.1109/TNNLS.2016.2514368
36. Gao H, Kuang L, Yin Y, Guo B, Dou K. Mining consuming behaviors with temporal evolution for personalized recommendation in mobile marketing apps. *Mob Netw Appl*. 2020;25:1233-1248.
37. Meng S, Xu J, Wang H, Yuan R, Zhang J, Li Q. Time-aware scalable recommendation with clustering-based distributed factorization for edge services. *World Wide Web*. 2021;1-19. doi:10.1007/s11280-021-00956-6
38. Bathla G, Aggarwal H, Rani R. Scalable recommendation using large scale graph partitioning with pregel and giraph. *Int J Cognit Inform Natural Intell (IJCINI)*. 2020;14(4):42-61.
39. Sim KM. Agent-based approaches for intelligent intercloud resource allocation. *IEEE Trans Cloud Comput*. 2019;7(2):442-455.
40. Zheng X, Xu LD, Chai S. QoS recommendation in cloud services. *IEEE Access*. 2017;5:5171-5177.
41. Akbulut A, Catal C, Akbulut FP. A cloud-based recommendation service using principle component analysis–Scale-invariant feature transform algorithm. *Neural Comput Appl*. 2017;28(10):2859-2868.
42. Liu J, Chen Y. A personalized clustering-based and reliable trust-aware QoS prediction approach for cloud service recommendation in cloud manufacturing. *Knowl-Based Syst*. 2019;174:43-56.
43. Concept drift-aware temporal cloud service APIs recommendation for building composite cloud systems. *J Syst Softw* 2021; 174: 110902.
44. Zhang L, Wei Q, Zhang L, Wang B, Ho WH. Diversity balancing for two-stage collaborative filtering in recommender systems. *Appl Sci*. 2020;10(4):1257.
45. Cloud Service Selection. State-of-the-art and future research directions. *J Netw Comput Appl*. 2014;45:134-150.
46. Chang C, Liu P, Wu J. Probability-based cloud storage providers selection algorithms with maximum availability. Proceedings of the 2012 41st International Conference on Parallel Processing; 2012:199-208.
47. Yang J, Lin W, Dou W. An adaptive service selection method for cross-cloud service composition. *Concurr Comput Pract Exp*. 2014;25(18):2435-2454.

48. Cai X, Hu Z, Chen J. A many-objective optimization recommendation algorithm based on knowledge mining. *Inf Sci*. 2020;537:148-161.
49. Altosaar J, Ranganath R, Tansey W. RankFromSets: scalable set recommendation with optimal recall. *Stat*. 2021;10(1):e363.
50. Wang S, Gong M, Wu Y, Zhang M. Multi-objective optimization for location-based and preferences-aware recommendation. *Inf Sci*. 2020;513:614-626.
51. Cai X, Hu Z, Zhao P, Zhang W, Chen J. A hybrid recommendation system with many-objective evolutionary algorithm. *Expert Syst Appl*. 2020;159:113648.
52. Moskalenko O, Parra D, Saez-Trumper D. Scalable recommendation of wikipedia articles to editors using representation learning; 2020. arXiv preprint arXiv:2009.11771.
53. Sahu P, Raghavan S, Chandrasekaran K. Ensemble deep neural network based quality of service prediction for cloud service recommendation. *Neurocomputing*. 2021;465:476-489.
54. Han J, Zheng L, Xu Y, et al. Adaptive deep modeling of users and items using side information for recommendation. *IEEE Trans Neural Netw Learn Syst*. 2019;31(3):1-12.
55. Zhang S, Liu H, He J, Han S, Du X. Deep sequential model for anchor recommendation on live streaming platforms. *Big Data Mining Anal*. 2021;4(3):173-182. doi:10.26599/BDMA.2021.9020002
56. Xu X, Li H, Xu W, Liu Z, Yao L, Dai F. Artificial intelligence for edge service optimization in internet of vehicles: a survey. *Tsinghua Sci Technol*. 2022;27(2):270-287. doi:10.26599/TST.2020.9010025
57. Tang W, Yan Z. CloudRec: a mobile cloud service recommender system based on adaptive QoS management. TRUSTCOM '15; 2015:9-16; IEEE Computer Society.
58. Lin D, Squicciarini AC, Dondapati VN, Sundareswaran S. A cloud brokerage architecture for efficient cloud service selection. *IEEE Trans Serv Comput*. 2019;12(1):144-157. doi:10.1109/TSC.2016.2592903
59. Yu Z, Wong RK, Chi C. Efficient role mining for context-aware service recommendation using a high-performance cluster. *IEEE Trans Serv Comput*. 2017;10(6):914-926.
60. Zhong Y, Fan Y, Huang K, Tan W, Zhang J. Time-aware service recommendation for mashup creation. *IEEE Trans Serv Comput*. 2015;8(3):356-368. doi:10.1109/TSC.2014.2381496
61. Zhang Q, Lu J, Wu D, Zhang G. A cross-domain recommender system with kernel-induced knowledge transfer for overlapping entities. *IEEE Trans Neural Netw Learn Syst*. 2019;30(7):1998-2012. doi:10.1109/TNNLS.2018.2875144
62. Wang C, Niepert M, Li H. RecSys-DAN: discriminative adversarial networks for cross-domain recommender systems. *IEEE Trans Neural Netw Learn Syst*. 2019;31(8):1-10. doi:10.1109/TNNLS.2019.2907430
63. Wang L, Zhang X, Wang T, et al. Diversified and scalable service recommendation with accuracy guarantee. *IEEE Trans Comput Soc Syst*. 2020;8(5):1-12.
64. Nitu P, Coelho J, Madiraju P. Improvising personalized travel recommendation system with recency effects. *Big Data Mining Anal*. 2021;4(3):139-154. doi:10.26599/BDMA.2020.9020026
65. Gong W, Zhang W, Bilal M, Chen Y, Xu X, Wang W. Efficient web APIs recommendation with privacy-preservation for mobile app development in industry 4.0. *IEEE Trans Ind Inform*. 2021;1-9. doi:10.1109/TII.2021.3133614
66. Qi R, Zhou J, Song X. An effective clustering method for finding density peaks. Proceedings of the 2018 IEEE International Conference on Parallel Distributed Processing with Applications (ISPA); 2018:39-46
67. Harper FM, Konstan JA. The MovieLens datasets: history and context. *ACM Trans Interact Intell Syst*. 2015;5(4):1-19. doi:10.1145/2827872
68. Liu NN, Meng X. Wisdom of the better few: cold start recommendation via representative based rating elicitation. Proceedings of the ACM Conference on Recommender Systems; 2011:37-44.
69. Zhang C, Li Z, Li T, et al. P-CSREC: a new approach for personalized cloud service recommendation. *IEEE Access*. 2018;6:35946-35956. doi:10.1109/ACCESS.2018.2847631
70. Zheng M, Min F, Zhang H, Chen W. Fast recommendations with the M-distance. *IEEE Access*. 2016;4:1464-1468.
71. Huang W, Li Q, Meng S. KG2Rec: LSH-CF recommendation method based on knowledge graph for cloud services. *Wirel Netw*. 2020;1-12. doi:10.1007/s11276-020-02387-z
72. Shani G, Gunawardana A. Evaluating recommendation systems. In: Ricci, F., Rokach, L., Shapira, B., Kantor, P. (eds) *Recommender Systems Handbook*; Boston, MA: Springer US. 2011:257-297.

How to cite this article: Qi R, Zhou J, Wang Z, Song X. An elastic recommender process for cloud service recommendation scalability. *Concurrency Computat Pract Exper*. 2022;e7066. doi: 10.1002/cpe.7066