Chapter 7
Towards Information Enrichment through Recommendation Sharing

Li-Tung Weng, Yue Xu, Yuefeng Li and Richi Nayak

Abstract Nowadays most existing recommender systems operate in a single organisational basis, i.e. a recommender system recommends items to customers of one organisation based on the organisation's datasets only. Very often the datasets of a single organisation do not have sufficient resources to be used to generate quality recommendations. Therefore, it would be beneficial if recommender systems of different organisations with similar nature can cooperate together to share their resources and recommendations. In this chapter, we present an Ecommerce-oriented Distributed Recommender System (EDRS) that consists of multiple recommender systems from different organisations. By sharing resources and recommendations with each other, these recommenders in the distributed recommendation system can provide better recommendation service to their users. As for most of the distributed systems, peer selection is often an important aspect. This chapter also presents a recommender selection technique for the proposed EDRS, and it selects and profiles recommenders based on their stability, average performance and selection frequency. Based on our experiments, it is shown that recommenders' recommendation quality can be effectively improved by adopting the proposed EDRS and the associated peer selection technique.

7.1 Introduction

Recommender systems are being applied in an increasing number of ecommerce sites to increase their business sales by helping consumers locate desired items to purchase. Generally, recommender systems make recommendations to users based on their implicit or explicit preferences, the preferences of other users, and item and user attributes [14, 13]. [3] suggested five different categories of recommender systems based on the information resources and the prediction algorithm employed.

Queensland University of Technology
Brisbane, QLD 4001, Australia

L. Cao (ed.), Data Mining and Multi-agent Integration, DOI: 10.1007/978-1-4419-0522-2_7,

© Springer Science + Business Media, LLC 2009

Among these five categories, collaborative filtering and content based filtering are the two most recognized and widely applied techniques.

Collaborative filtering based methods [7, 19] take the preferences of users (e.g. user ratings and purchase histories) as the major information resources in order to aggregate opinions from users with similar preferences, and the recommendations are generated based on these aggregated opinions. On the other hand, content-based filtering methods use information retrieval related techniques to recommend items that have similar contents (or attributes) to the user preferred items.

One of the most well-known challenges for recommender systems is the cold-start problem. The cold-start problem occurs when making recommendations for new users with their preferences unknown (i.e. lack of previous rating information or transaction histories), or suggesting new items that no one has yet rated or purchased [15]. Collaborative filtering based recommenders are very vulnerable to the cold-start problem because they operate solely on the basis of the user preference information, and therefore many works propose the so called "hybrid recommenders" that combine both content-based filtering and collaborative filtering together to replenish the insufficient user preference information [3, 16].

Even though hybridization based recommenders have been widely applied against the cold-start problem, they are still not comparable to using non-hybrid recommenders with sufficient information resources [3]. For instance, assuming a new ecommerce site wants to run a recommender system with very limited user records and product catalogue in its database, it will not be able to generate quality recommendations despite the hybridization techniques are employed.

In this chapter, a novel strategy for alleviating the cold start problem is explored. The basic idea of the strategy is to increase data volume of recommenders via allowing them to share and exchange recommendations with each other over a distributed environment. As mentioned previously, most of the existing recommender systems are designed for one single organisation (i.e. business to customer (B2C) recommenders), and in general, one single organisation may not possess sufficient information or data for analysis in order to give their customers precise and high quality recommendations. Therefore, it can be beneficial if organisations can share their information resources (i.e. products and customer database) and recommendations boundlessly (i.e. build recommendation systems at an interorganisational level).

This chapter presents a framework for distributed recommendation sharing among recommenders, namely Ecommerce-oriented Distributed Recommender System (EDRS). The proposed EDRS is different from existing distributed recommender systems. While existing distributed recommender systems are mainly designed for C2C (Customer to Customer) based applications (such as file sharing applications), the proposed EDRS introduces additional B2B (Business to Business) features on top of the standard B2C recommender systems. Specifically, the goal of the EDRS is to allow the standard recommenders from existing ecommerce sites or e-shops (e.g. Amazon.com, Netflix.com) to improve their recommendation quality towards their users by sharing their information resources and recommendations with each other.

7.2 Prior and Related Work

Notwithstanding the popularity of centralised recommenders in last decades, recommender systems that operate on distributed environments or decentralised infrastructures have started to attract attention from researchers, and these systems are commonly referred to as distributed recommender systems or decentralised recommender systems [4, 10].

Generally, a distributed recommender system associates each of its users with a recommender agent (or peer recommender) on his or her personal computer (client-side machine). These recommender agents gather user profile information from their associated users, and exchange these profile information with other agents over a distributed network (e.g. internet), in the end a recommender agent makes recommendations to its associated user by utilizing the user's personal profile as well as these gathered peer profiles (i.e. profiles of other users gathered from other recommender agents) [17, 18].

There are several reasons that lead to increasing popularity of distributed recommender systems:

- The fast growing development of internet related technologies and applications (e.g. the Grid, ubiquitous computing, peer-to-peer networks for file sharing and collaborative tasks, Semantic Web, social communities, WEB 2.0, etc.) has yielded a wealth of information and data being distributed over most of nodes (i.e. web server, personal computer, mobile phone, etc.) in the internet. Hence, getting information recommended from only one single source (e.g. ecommerce site) is no longer sufficient for many users, and instead, they are thirsty for richer information from multiple sources [17]. For example, the peer-to-peer (P2P) based file sharing protocol, BitTorrent (www.bittorrent.com), has proven to be among the most competent methods to allow large numbers of users to efficiently share large volumes of data. Instead of storing files or data in a central file server (e.g. FTP server), BitTorrent stores files in multiple client machines (i.e. peers), and when a file is requested by a user (i.e. a peer), the user can download this file simultaneously from multiple peers [4]. Intuitively, as there is no central server for storing file contents and user (or peer) profiles in BitTorrent, distributed recommender systems would be more suitable to be applied to such system than centralised recommenders.
- User privacy and trust is another area that distributed recommender systems are considered superior to centralised recommender systems. In a centralised recommender system, all user information and profiles are possessed by the ecommerce site that runs the recommender system, and this can result in privacy and trust concerns. Firstly, a centralised recommender system might share users' personal information and profile in inappropriate ways (e.g. selling user information to others), and the users generally have no control over it. Secondly, a centralised recommender system owned by an ecommerce site might make recommendations for the business's own good instead of serving users' needs. For example, a

site can adjust its recommender's configuration, so it only recommends products that are overstock instead of required by the users.

- The privacy and trust issues are alleviated by distributed recommender systems. In a distributed recommender system, users' personal information and profiles are stored in their own machines, and they generally can explicitly define and set which parts of their personal data and profiles are sharable. In addition, because a recommender agent in a distributed recommender system is a piece of software that runs independently on each client's machine and it usually gathers information only from other peer agents rather than from an ecommerce site, therefore, it is less possible that the ecommerce sites can manipulate the recommendations to the users [12].
- In addition, scalability is one of the major challenges for the centralised recommender systems. It is because correlating user interests in a large dataset can be very computationally expensive (it normally require a quadratic order matching steps). Some research works, therefore, suggest implementing recommender systems in a decentralised fashion to improve the scalability and computation efficiency [12].

Most existing works on distributed recommender systems are mainly designed for peer-to-peer (P2P) or file sharing applications (which usually adhere to C2C paradigm). Awerbuch's [1] work provides a generalized view to these distributed recommenders. Awerbuch suggested a formalized model for the C2C distributed recommender systems. In Awerbuch's model, for the distribute system with m users and n items, there will be m recommender systems (i.e. agents or peers), and each of the recommender agents will associate with exactly one user. Each recommender works on behalf of the associated user either to trade recommendations with other agents or probe the items on its own. Each recommender aims to finally discover the p items preferred by the associated user, where $p \leq n$. In Awerbuch's opinion, from the perspective of the entire distributed recommender system, the goal is rather similar to the "matrix reconstruction" proposed by Drineas et al. [6]; the overall task is to reconstruct an $m \times n$ user preference matrix in a distributed fashion. It can be observed that many distributed recommender systems belong to this model.

Generally, the goal of these C2C based distributed recommenders is to avoid central server failure and protect user privacy (no central database containing information about customers) [1, 18, 16] . However, most of them are not aiming at improving their effectiveness or the recommendation quality. By contrast, the goal of the proposed EDRS is aiming at improving the recommendation quality and alleviating the cold start problem. Hence, the infrastructure of the proposed distributed recommender system is different from Awerbuch's model as well as many other existing systems. EDRS contains a set of classical recommenders, and each of them serves their own set of users. Our goal is to improve the recommendation quality of these recommenders by allowing them making recommendations for others in a decentralised fashion. Thus, for the profiling and selection problem, we proposed a more sophisticated strategy rather than random sampling for recommender peers to explore others.

Moreover, recommender systems and information retrieval (IR) systems are generally considered similar research fields [13], since both of them try to satisfy users' information needs by either retrieving the most relevant documents or recommending the most preferred items to users. Information retrieval retrieves documents based on users' explicit queries, while recommender systems recommend items or products based on users' previous behaviour. In distributed IR [2, 5] , the entire document collection is partitioned into subcollections that are allocated to various provider sites, and the retrieval task then involves:

- Querying minimal number of subcollections (to improve the efficiency), and ensure the selected subcollections are significant to uphold the retrieval effectiveness.
- Merging the queried results (fusion problem) that incorporates the differences among the subcollections in such a way that no decrease in retrieval effectiveness is effectuated with respected to a comparable non-distributed setting.

For distributed recommender systems, the recommender peer selection and recommendation merging are also two important tasks. In fact, one of the major focuses of the works presented in this chapter is to design an effective recommender peer profiling and selection strategy. The selection criteria for distributed IR including the: efficiency (selecting minimal number of subcollections) and effectiveness (retrieving the most relevant documents) is similar to the criteria for the proposed distributed recommender system. However, in distributed IR, the collection selection is content based [5] and it requires the subcollections provide or use sampling techniques to get subcollection index information (eg. the most common terms or vocabularies in the collection) and statistical information (eg. document frequencies). By contrast, the proposed selection technique requires no content related information about recommender peers (assuming recommender peers share minimal knowledge to each other), the proposed selection algorithm is based on the observed previous performance (i.e. how well a recommender peer's recommendations satisfy the users) about each of the recommender peers.

7.3 Ecommerce-Oriented Distributed Recommender System

As mentioned earlier, the goal of the proposed distributed recommender system is to allow standard recommenders to overcome cold-start problem and improve recommendation quality by cooperating, interacting and communicating with recommenders of other parties (e.g. other ecommerce sites). Hence, the proposed system is designed to contain of a set of recommenders from different sites and each of these recommenders is associated with their own users. Note, it is possible that a user might visit multiple sites, and therefore two or more recommenders may share common users. Similar to the centralised paradigm, each recommender peer in the proposed system still serve its own users in a centralised fashion (i.e. the recommender stores all its user and product data in a central place within the recommender). How-

ever, in the proposed system, the recommender peers can enrich their information sources by communicating and cooperating with each other. A general overview of the proposed system is depicted in Fig.7.3.

Because the proposed distributed recommender system is designed to benefit ecommerce sites (rather than focusing on helping users to gain more controls on recommenders), we therefore name our system as "Ecommerce-oriented Distributed Recommender System", and abbreviate it to EDRS. We also abbreviate the standard Distributed Recommender System to DRS and Centralised Recommender System to CRS in order to clarify and differentiate the three different system paradigms.

Before explaining the proposed distributed recommender framework in more detail, some general differences among the EDRS, DRS and CRS are investigated. In particular, these systems are compared according to the following aspects:

- **Ecommerce Model:** based on the general ecommerce activities and transactions involved in the recommenders' host application domains, we can roughly categorize them into three different models, namely, Business-to-Business (B2B), Business to Customer (B2C) and Customer to Customer (C2C). In B2B model, activities (e.g. transactions, communications, interactions, etc.) mainly occur among businesses. In the B2C model, activities are mainly between businesses and customers, and the most typical example is activities of E-businesses serving end customers with products and/or services. Finally, the C2C model involves the electronically-facilitated transactions between consumers. A typical example is the online auction (e.g. eBay), in which a consumer posts an item for sale and other consumers bid to purchase it.
- **Architectural Style:** an architectural style describes a system's layout, structure, and the communication of the major comprising system modules (or software components). Over past decades, many architectural styles have been proposed, such as, Client-Server, Peer-to-Peer (P2P), Pipe and Filter, Plugin, Service-oriented, etc. Client-Server and Peer-to-Peer are the two major architectural styles related to our work, and therefore will be explained in more details. The Client-Server architecture usually consists of a set of client systems and one central server system, client systems make service requests over a computer network (e.g. internet) to the server system, and the server system fulfils these requests. Peer-to-Peer architec-ture consists of a set of peer systems interacting with each other over a computer network, and it does not have the notion of clients and servers, instead, all peer systems operate simultaneously as both servers and clients to each other.
- **Communication Paradigm:** based on how two types of entities communicate with each other within a system, three major communication paradigms have been proposed, and they are: One-to-One, One-to-Many and Many-to-Many communication paradigms (or relationships). In One-to-One communication paradigm, communication occurs only between two individual entities, example applications include: e-mail, FTP, Telnet, etc. By contrast, a website that displays information accessible by many users is considered having a One-to-Many relationship. In Many-to-Many paradigm, entities communicate freely

with many others, example applications include: file sharing (multiple users to multiple users), Wiki (multiple authors to multiple readers), Blogs, Tagging, etc.

Figure 7.1 shows a general overview of a standard centralised recommender system (i.e. CRS). The host application of CRS is usually an ecommerce site (e.g. Amazon.com, Netflix.com, etc.) which possesses all user/product relevant information, and the recommender then utilizes all the information from the site to make personalized recommendations to the site's users and further create business values to the ecommerce site. As the nature of the CRS is to serve the users (i.e. customers) and to satisfy the users' information needs to the ecommerce site (i.e. business), it can be considered as adhering to the B2C paradigm. It is usually implemented based on the Client Server architecture because the entire recommendation generation process occurs only within the central server, and users interact with the recommender though thin clients (e.g. web browsers) whose major functions are presenting users the recommendations generated from the server and sending users' information requests to the server. In the most common case, all users of a site are served by a single recommender, therefore, the communication paradigm between recommenders and users in CRS is considered as One-to-Many.

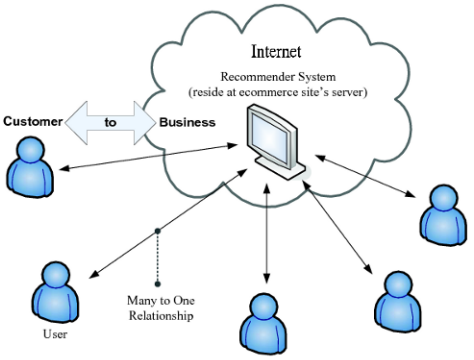

Fig. 7.1: Classical centralised recommender system

The standard distributed recommender system (DRS), as depicted in Fig. 7.2, differs from CRS in all of the three mentioned aspects. First of all, it emphasizes users' privacy protection by preventing personal user data being gathered and used (or misused) by ecommerce site owners (or businesses), hence adheres to the Customer-to-Customer model (as Business entities are evicted from the system for privacy protection). It is shown in Fig. 7.2 that, a standard distributed recommender system associates every user in the system with a recommender peer serving the user's personal information needs, hence the relationship between the user and recommender peer is considered as One-to-One. On the other hand, in order to make better recommendations to its user, a recommender peer might need to communicate with other peers to exchange its user's data (in a privacy protected way) with other peers or to get recommendations from other peers because there is no central place for storing

all users' data. The relationship among recommender peers in the DRS is considered as Many-to-Many, as a peer can both communicate to and be communicated by many other peers. Finally, because all recommender peers are equipped with similar set of functionalities (i.e. gather information from others and making recommendation to its user) and operate independently and autonomously from others, therefore they are commonly modelled and implemented using the Peer-to-Peer architectural style.

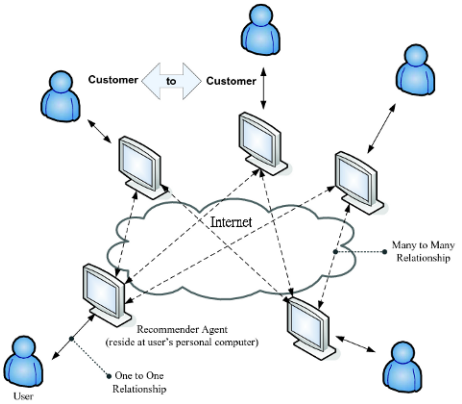

Fig. 7.2: Standard distributed recommender system

The proposed Ecommerce-oriented Distribute Recommender System (EDRS) (depicted in Fig. 7.3), can be thought as a combination of the two systems (centralised recommender and DRS) described above. Similar to the DRS, EDRS consists of a set of recommender peers and a set of users. However, while one user is associated with exactly one recommender peer in the standard distributed recommender system, the proposed system can be considered as a set of centralised recommender systems cooperate together to serve their own set of users, and therefore each recommender peer needs to interact (i.e. make recommendations to) with multiple users. Moreover, it is also possible that in our system a user is associated with more than one recommenders (i.e. he or she can visit multiple sites); for instance, a book reader might try to find a book in both Amazon.com and Book.com. Because a recommender peer in our system can serve multiple users and a user can make recommendation requests to multiple recommender peers, the relationship between users and recommender peers is considered as Many-to-Many. As mentioned previously, the recommender peers in EDRS might interact and cooperate with each other to improve their recommendation quality, and hence, apart from the Many-to-Many relationship between users and recommender peers, another Many-to-Many communication relationship exists among the peers.

Because EDRS is still designed for normal ecommerce sites, such as e-book stores like Amazon.com, its major ecommerce model is therefore same as CRS, that is, Business-to-Customer. Besides, since EDRS introduces additional commu-

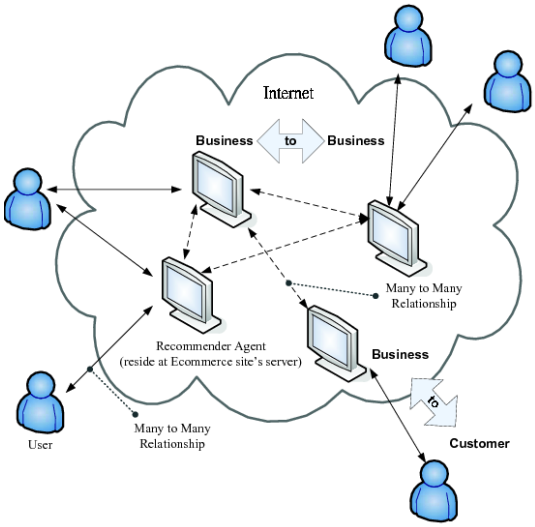

Fig. 7.3: Proposed distributed recommender system

nication and cooperation for recommenders of different sites, it is expected that the cooperation of these recommenders (also their sites) will confirm to the Business-to-Business based model.

The implementation of the proposed EDRS involves both Peer-to-Peer and Client-Server architectural styles. Client-Server architecture is employed to model a recommender peer (i.e. the server) and its users (i.e. the clients). Similar to the centralised recommender, the entire recommendation generation process is done by the recommender situated at the server side, and the users make requests to the recommender through thin clients such as web browsers. The architectural style for the network among the recommender peers is modelled with Peer-to-Peer architecture. As mentioned before, Peer-to-Peer based architecture assumes that the peers are independent and autonomous from each other, and especially they should be loosely coupled. Such definition is suitable for modelling the relationship between the recommender peers' host sites, as they are both logically and physically independent and autonomous from each other (as they are different e-commerce sites and organisations). While both DRS and the proposed EDRS can be modelled with the Peer-to-Peer architecture, the recommender peers in EDRS are more strongly coupled together than in standard DRS, because the recommender peers in EDRS need to gather/distributed information and suggestions from/to each other in a timely and effective fashion to achieve their common goal (i.e. satisfy their users' information and recommendation needs).

To the best of our knowledge, the concept of the proposed EDRS has not yet been mentioned and investigated by other works. Also, it is different from existing recommender systems (both centralised and distributed ones) at several high level aspects.

7.3.1 Interaction Protocol

As mentioned earlier, the interaction, communication and cooperation of the recommender peers in the proposed EDRS can be modelled with the Peer-to-Peer based architectural style. In particular, the ”Contract Net Protocol” (CNP) is employed as the foundation for modelling the system, which provides the basis for coordinating the interaction and communication among the recommender peers. Contract Net Protocol is a high level communication protocol and system modelling strategy for Peer-to-Peer architectural based systems (or other distributed systems) [9, 20] Weiss, 1999. In CNP, peers in the distributed system are modelled as nodes and the collection of these nodes is referred to as a contract net. In CNP based systems, the execution of a task is dealt with as a contract between two nodes, each node plays a different role, one of them is the manager role and the other is the contractor role. The role of a manager is responsible for monitoring the execution of a task and processing the results of its execution. On the other hand, the role of a contractor is responsible for the actual execution of the task. Note, the nodes are not designated a priori as contractors or managers, rather, any nodes may take on either roles dynamically based on the context of their interaction and task execution [20]. A contract is established by a process of mutual selection based on a two-way transfer of information. In general, available contractors evaluate task announcements made by managers and submit bids on those for which they are suited. The managers evaluate the bids and award contracts to the nodes (i.e. contractors) that they determine to be most qualified [20].

In the case of the proposed EDRS, the recommender peers are modelled as the nodes in the contract net. Depending on difference circumstances, each recommender peer plays manager role and contractor role interchangeably. When a recommender peer makes requests for recommendations to other peers, it is considered as a manager peer. On the other hand, the recommender peer that receives a request for recommendations and provides recommendations to other manager peers is considered as a contractor peer. The roles of the manager peer and the contractor peer and their interactions are depicted in Fig. 7.4.

The communication steps involved in the interaction are indicated by the numbers in Fig. 7.4 and explained as follows:

1. User sends a request for recommendations. The recommender peer who received the request and is responsible for making the recommendation to the user is considered as in manager role.
2. Based on the user’s request and profile, the manager peer selects suitable peer recommenders to help it on making better recommendations to the user.
3. The manager peer makes requests to the peers for recommendation suggestions. The request message may only contain the user’s item preferences (i.e. the user’s rating data); however the identity of the user is remain anonymous for privacy protection.
4. Each contractor peer generates recommendations based on the received request.

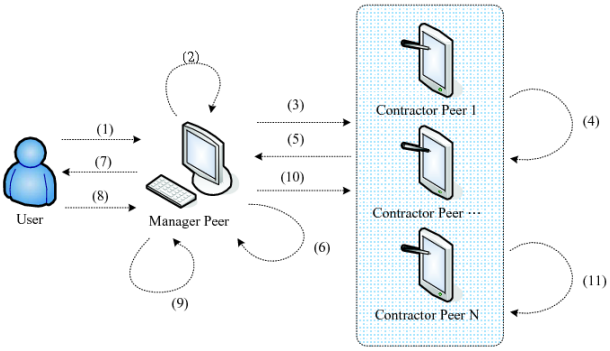

Fig. 7.4: High level interaction overview for EDRS (based on contract net protocol)

5. The contractor peers send back their recommendation suggestions to the manager peer.
6. After the manager peer received the suggestions from the contractors, it then synthesizes and merges these recommendation suggestions.
7. Based on the synthesized recommendation suggestions from the contractor peers (might also include the manager peer's own recommendations) the manager peer generates the item recommendations to the user.
8. When the user received the recommendations, he or she might supply implicit or explicit ratings to the recommendations. That is, the user might provide indications about whether he or she likes or dislikes one or more items in the recommendation list.
9. Based on the user rating feedbacks, the manager peer can objectively evaluates each of the peers' (i.e. contractors') performances to the recommendation suggestions they supplied and update its profiles about these peers.
10. The manager peer sends feedbacks and rewards to the contractor peers based on their performances to the task.
11. When the contractor peers received feedbacks about the performances of their recommendation suggestions, they then update their profiles about the manage peer in order to improve their future suggestions.

From Fig. 7.4, it can be seen that when a recommender peer is requested to make recommendations for a user, it acts as a manager peer. In the role of a manager peer, the recommender first generates a strategy about how and what to recommend to the user based on the user's profile and request, then chooses a set of recommender peers (in this context, they act as contractor peer) based on the profiles of peer recommenders, and finally makes requests for recommendations to these selected contractor peers. When these selected contractor peers received the requests, they then construct and return their recommendation suggestions based on the requests received and the manager peer's profile (e.g. preferences, domain of interests, trustworthiness and etc.). After the manager peer received the recommendations returned from the contractor peers, it then merges the recommendations (also include its own

recommendations) and returns to the user. According to the recommendations received from the manager peer, the user might either explicitly or implicitly give feedbacks or ratings about the recommendations to the manager peer. After receiving the user's feedback, the manager peer will evaluate the performance of each of the selected contractor peers, update its profiles about them, and then construct the feedbacks and make rewards to the contractor peers. Finally, the contractor peers will update theirs profile about the manager peer based on the given rewards and feedbacks.

In order to carry out the proposed interaction described above, tasks such as *recommender peer selection*, *recommendation generation*, *recommendation merge*, *peer feedback and profile update* will need to be considered. Among all these formentioned tasks, recommender peer profiling and selection is the major focus of this chapter, and a novel contractor peer profiling and selection strategy is proposed, discussed and investigated in Sect. 7.4.

7.4 Peer Profiling and Selection

Part of the major contributions in this chapter includes a recommender profiling scheme (for manager peers to profile contractor peers) and a recommender selection algorithm designed for the proposed EDRS. In particular, the recommender peer selection problem is modeled as the classical exploitation vs. exploration (or k-armed bandit) problem [11], in which the recommender selection for the manager peer has to be balanced between choosing the best known contractor peers to keep users satisfied and selecting other unfamiliar contractor peers to obtain knowledge about them. The proposed recommender selection algorithm is based on evaluating the Gittins Indices [11] for every recommender peer, and the indices reflect the average performance, stability and selection frequency of the recommenders (i.e. contractor peers).

7.4.1 System Formalization for EDRS

We envision a world with a set of users and items, and they are denoted by $U = \{u_1, u_2, ..., u_n\}$ and $T = \{t_1, t_2, ..., t_m\}$ respectively. The proposed distributed recommender system (EDRS) denoted as Φ contains a set of l recommender peers $\phi_1, \phi_2, ..., \phi_l$, i.e. $\Phi = \{\phi_1, \phi_2, ..., \phi_l\}$. The number of recommender peers is much smaller than the number of users in our system, i.e. $l \ll n$. Each recommender peer $\phi_i \in \Phi$ has a set of users denoted as $U_i \in U$, and a set of items denoted as $T_i \in T$, where $U = \bigcup_{\phi_i \in \Phi} U_i$ and $T = \bigcup_{\phi_i \in \Phi} T_i$. It should also be noted that some users and items can be owned by more than one recommender peers.

7.4.2 User Clustering

Intuitively, a large set of users can be separated into a number of clusters based on the user preferences. Since users within the same cluster usually share similar tastes [3] and a cluster with a large number of users and a high degree of intra-similarity can better reflect the potential preferences of the users belonging to the cluster, a collaborative filtering based recommender can improve its recommendation quality by searching similar users within clusters rather than the whole user set [13]. However, different user clusters often vary in quality. The performance of such clustering based collaborative filtering system is strongly influenced by the quality of the clusters [13]. For a given recommender, some users might be able to receive better recommendations if they belong to a cluster with better quality (the cluster has a large number of users and a high intra-similarity), whereas some other users may not be able to get constructive recommendations because the cluster to which they belong is small and has a low intra-similarity. This situation is closely related to the cold-start problem [15] which happens when a recommender makes recommendations based on insufficient data resources. Therefore, even for the same recommender, the recommendation performance might be different for different clusters of users if different user clusters have different quality. In order to provide good recommendations to various users, the proposed EDRS allows its recommender peers (i.e. manager peers) to choose peers (i.e. contractor peers) for recommendations to the current user based on their performances to a particular user cluster to which the current user belongs. We expect this design to solve the cold start problem because a recommender which is making recommendations to a user who belongs to a weak cluster can get recommendations from recommender peers who have performed well to that group of users.

In the proposed EDRS, every recommender peer has its own set of user clusters, and we denote the set of user clusters owned by $\phi_i \in \Phi$ as $UC_i = \{uc_{i,1}, uc_{i,2}, ..., uc_{i,m_i}\}$, such that $uc_{i,j} \subseteq U_i$. In addition, for the simplicity of the system, all user clusters are assumed to be crisp sets. Because different recommender peers have different user sets and different clustering techniques, the size of their cluster set might vary as well.

7.4.3 Recommender Peer Profiling

In this section, we present our approach to profile the recommender peers within the proposed EDRS. To begin with, the performance evaluation of the recommender peers is explained. The performance of a recommender peer is measured by the degree of user satisfactory to the recommendations made by the recommender [8]. In our system, a recommender peer ϕ_i makes recommendations to a user with a set of k items $P_i = \{p_{i,1}, p_{i,2}, ..., p_{i,k}\}$ where $P_i \subseteq T_i$. Once having received the recommendations, the user then input his or her evaluations to each of the k items. We use r_a to denote the user's rating to item $p_{i,a} \in P_i$. The value of r_a is between 1 and 0 which

indicates how much the user likes item $p_{i,a}$. When r_a closes to 1, it indicates the user highly prefers the item, by contrast when r_a closes to 0, the user dislikes the item. Hence, each time a recommender peer generates a recommendation list (eg. P_i) to a user, it will get feedback $R = \{r_1, r_2, ..., r_k\}$ from the user, where $r_a \in (0,1)$. With R, we can compute the recommender peer's current performance χ to the user by:

$$\chi = \frac{\sum_{r \in R} r}{|R|} \tag{7.1}$$

Equation (7.1) measures the current performance of a recommender peer to a particular user in the current recommendation round. We can use the average performance of the recommender to the users in the same cluster to measure its performance to this group of users. The average performance measures how well the recommender averagely performed in the past. However, the average performance doesn't reflect whether the recommender is generally reliable or not. Hence, we employed the standard deviation to measure the stability of the recommender. Another factor that should be taken into account for profiling a recommender is the selection frequency which indicates how often the recommender has been selected before. In our system, we profile each recommender peer from the three aspects: recommendation performance, stability, and selection frequency. As mentioned previously, a recommender will seek for recommendations from other peers when it receives a request from a user. Broadcasting the user request to all peers is one solution, but obviously it is not a good solution since not all of the peers are able to provide high quality recommendations. In EDRS, the recommender peers (i.e. manager peers) will select the most suitable peers (i.e. contractor peers) for recommendations based on their profiles. Therefore, each recommender peers in EDRS keeps profiles to each of the other recommender peers.

A recommender peer may perform differently to different user clusters. Therefore its performances to different user clusters are different. For recommender $\phi_i \in \Phi$ which has m_i user clusters, that is, $UC_i = \{uc_{i,1}, uc_{i,2}, ..., uc_{i,m_i}\}$, we use $Q^i_{j,h}$ to denote the average performance of peer $\phi_j \in \Phi$ to ϕ_i's user cluster $uc_{i,h}$. Hence, we can use a $z \times m_i$ matrix $Q^i = \{Q^i_{j,h}\}_{zm_i}$ to represent the average performance of each of the other peers to each of ϕ_i's user clusters, where $z = |\Phi| - 1$ and $m_i = |UC_i|$. Q^i is called as the peer average performance matrix of ϕ_i. Similarly, we use S^i and F^i to represent the stability and selection frequency of other peers to ϕ_i. $S^i = \{S^i_{j,h}\}_{zm_i}$ and $F^i = \{F^i_{j,h}\}_{zm_i}$ are called as the peer stability matrix and peer selection frequency matrix respectively. In summary, a recommender ϕ_i's peer profile is defined as $\mathbb{P}^i = \{Q^i, S^i, F^i\}$ which consists of the three matrixes representing peer recommender's average performance, stability, and selection frequency, respectively.

Initially, the Q^i, S^i and F^i of ϕ_i are all zero matrixes, because ϕ_i has no knowledge about other peers. These matrixes will be updated when a recommender peer ϕ_j helped ϕ_i (i.e. ϕ_j is in contractor role and ϕ_i is in manager role) to make a recommendation P_i for a user belonging to (or being classified to) a ϕ_i's user cluster $uc_{i,h}$. Suppose that R_j is the recommendation list returned by ϕ_j. Ideally, R_j is expected to be a subset of T_i. But usually $R_j \nsubseteq T_i$ since ϕ_i and ϕ_j may have dif-

ferent item sets. In the proposed EDRS, only the items which are in T_i are considered by ϕ_i. Let P_i be the final recommendation list made by ϕ_i to the user and $P_j = \{t | t \in R_j \cap T_i$ and selected by $\phi_i\}$ be the recommendation list made by ϕ_j and selected by ϕ_i during the merging process (the major focus of this selection is on peer profiling, other aspects of the proposed EDRS such as merging recommendations from different peers will be explained in latter sections). P_j should be a subset of P_i. After the recommendation P_i is provided to the user, ϕ_i will get a feedback list (i.e. the actual user ratings to the recommended items) R about P_i from the target user. With the user feedback R, Equation 7.2 will be used to compute ϕ_j's performance χ for the recommendation of this round (only the items in P_i are taken into consideration when compute the χ for ϕ_j) which is ϕ_i's observation about ϕ_j's performance to user cluster $uc_{i,h}$. The methods for updating the average quality, stability and selection frequency in ϕ_i's peer profile $\mathbb{P}^i = \{Q^i, S^i, F^i\}$ are given below, where $\tilde{Q}^i_{j,h}$, $\tilde{S}^i_{j,h}$, $\tilde{F}^i_{j,h}$ are the updated value for peer ϕ_j and cluster $uc_{i,h}$ in the three matrixes, respectively:

$$\tilde{Q}^i_{j,h} = \frac{Q^i_{j,h} \times F^i_{j,h} + \chi}{F^i_{j,h} + 1} \tag{7.2}$$

$$\tilde{F}^i_{j,h} = F^i_{j,h} + 1 \tag{7.3}$$

$$\begin{aligned} \tilde{S}^i_{j,h} &= 0 && \text{, if } F^i_{j,h} < 2 \\ &= \sqrt{\frac{[(F^i_{j,h}-1)\times S^i_{j,h}]^2 + \frac{(\chi - Q^i_{j,h})^2}{F^i_{j,h}+1}}{F^i_{j,h}}} && \text{, otherwise} \end{aligned} \tag{7.4}$$

7.4.4 Recommender Peer Selection

In this section, a novel technique is proposed that allows manager peers to effectively and efficiently select contractor peers based on the proposed recommender peer profiles described in Section 7.3 for assistances in making quality recommendations. The proposed peer selection strategy is based on the famous Gittins Indices technique [11] developed for solving the exploitation vs. exploration problem, as such, it enables the manager peers to efficiently learn their contractor peers as well as maintain their recommendation quality to the users.

7.4.4.1 Gittins Indices

The Gittins indices [11] is developed for the k-armed bandit problem (which is a subset of the exploitation vs. exploration problem) that deals with a slot machine with k arms. An amount of reward will be given when an arm is pulled. However, in each time period, only a limited number of arms can be pulled (normally one

arm). Different arms have different reward distributions, and the reward distributions for the arms are initially unknown. The objective is to choose which arms to pull that will maximize the total rewards over time based on previous experience and obtained rewards as well. Formally, the k-armed bandit problem is to schedule a sequence of pulls maximizing the expected present values of

$$\sum_{t=1}^{\infty} \alpha^t R(t) \tag{7.5}$$

where t indicates the time points, $R(t)$ denotes the sum of the rewards obtained by pulling a set of arms at t, and α is a fixed discount factor where $0 < \alpha < 1$.

Traditionally, dynamic programming was the preferred framework for solving the bandit problem. It requires analysis of all possible combinations of the pulling sequences. However, Gittins has developed a solution in 1972 that requires computation only on the current states of the individual arms. Gittins suggests comparing each potential action (i.e. a pull) against a reference arm with a known and constant reward instead of to compare all possible actions against each other [11]. Gittins proved it is optimal to select actions with expected rewards equal to the reference actions with the highest equivalent rewards (i.e. Gittins index values) for each pull [11].

Specifically, a Gittins index value of an arm is computed based on the average and standard deviation of the rewards generated from the arm as well as the number of times the arm has been pulled. The application of the Gittins indices for solving the multi-armed bandit problem is therefore straight forward: we simply compute the Gittins index values for every arms (based on their current average and standard deviation of the rewards generated and the number of times each of them are pulled), and pull the arm with the highest index value. As the arm selection task involves only the current states of the arms (i.e. current average and standard deviation of the rewards and number of the times being pulled), it is therefore both memory and computationally efficient (when comparing to dynamic programming based solutions).

Note, the theorem background and the relevant index value generation techniques of the Gittins Indices technique are detailed in [11], this work mainly focuses on the application of the Gittins indices in the context of the recommender peer selection task.

Given an arm which has been pulled for n times, and generated an average reward $\bar{x}$ with a standard deviation $\hat{s}$, Gittins denotes the index value for the arm as $v(\bar{x}, \hat{s}, n)$, and he also proved that:

$$v(\bar{x}, \hat{s}, n) = \bar{x} + \hat{s} \times v(0, 1, n) \tag{7.6}$$

where $v(0, 1, n)$ is the index value for an arm being pulled for n times with a zero average reward and a standard deviation of 1. Gittins has calculated the value of $v(0, 1, n)$ for different combination of α and n in [11]. Gittins suggested that by selecting the arms with the highest index value (i.e. (7.6)) in every selection round, the overall accumulated total reward can be optimized.

7.4.4.2 Selection Strategy for EDRS

Based on Sect. 7.3 when a manager peer ϕ_i wants to find a best contractor peer ϕ_i to make a recommendation to a user $u \in uc_{i,h}$ the following equation is used to select the most suitable peer:

$$\phi_j = \text{argmax}_{\phi_j \in \Phi \ \{\phi_i\}} Q^i_{j,h} + S^i_{j,h} \times v(F^i_{j,h}) \tag{7.7}$$

where$v(F^i_{j,h})$ is the Gittins index function that maps $F^i_{j,h}$ (i.e. selection frequency) to the corresponding $v(0,1,F^i_{j,h})$. In (7.7), ϕ_i firstly calculates the average performance, stability and selection frequency of the available peers to the user cluster that u belongs to (i.e. $uc_{i,h}$). Then ϕ_i computes the index values for every peers based on (7.7). Finally, the most preferred peerϕ_j will be the one which has the highest index value. By setting up a cutoff for the index value, multiple recommender peers with index values higher than the cutoff can be selected.

7.5 Experiments and Evaluation

In this experimentation, multiple recommenders with different capability in making recommendations are constructed, and we allow them to interact with each other based on the proposed EDRS framework. Essentially, these recommenders employ the proposed peer profiling and selection strategy presented in Sect. 7.4 to learn from and select each other in order to improve their recommendation making. Our main focuses are to examine whether incorporating helps from other recommenders can indeed improve recommenders' recommendation quality and also evaluate the effectiveness of the proposed profiling and selection strategy.

7.5.1 Data Acquisition

In this work, the "Book-Crossing" dataset (http://www.informatik.unifreiburg.de/ cziegler/BX/) is chosen to conduct the experiments. The "Book-Crossing" dataset is collected by Cai-Nicolas Ziegler in a 4-week crawl (August / September 2004) from the Book-Crossing community (http://www.bookcrossing.com/) with kind permission from Ron Hornbaker, CTO of Humankind Systems. It contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit / implicit) about 271,379 books. In the user ratings, 433,671 of them are the explicit user ratings, and the rest of 716,109 ratings are implicit ratings. The book taxonomy and book descriptors for the experiments are obtained from Amazon.com. Amazon.com's book classification taxonomy is tree-structured (i.e. limited to "single inheritance") and therefore is perfectly suitable to the proposed tech-

nique. The average number of descriptors per book is around 3.15, and the taxonomy tree formed by these descriptors contains 10,746 unique topics.

7.5.2 Experiment Setup

As the main purpose of this experiment is to evaluate the proposed interaction protocol and the peer profiling and selection technique (rather than evaluating a new recommendation technique or algorithm) in a distributed recommender system, therefore the overall setup of this experiment is different from the setup for non-distributed recommender systems.

In this experiment, it is required to simulate the interactions (i.e. profiling and selection) among the recommenders from different organisations, and therefore the first step in the experiment setup process is to construct multiple recommenders with different capabilities and underlying knowledgebase (i.e. datasets). Next, the testing dataset is constructed for evaluating the recommenders' recommendation quality. Importantly, the recommendation quality comparison between recommenders utilizing the proposed EDRS framework (i.e. getting helps from other recommenders) and standalone recommenders (i.e. making recommendations based on their own efforts) are carried out. Moreover, the effectiveness of the proposed peer profiling and selection technique is also examined by comparing it with other peer selection strategies. Note, the proposed peer profiling strategy requires the manager peers to get user feedbacks for all of their recommendations so they can determine their contractor peers performances based on the feedbacks and then update their peer profiles. Hence, it is necessary to provide a way to allow the user feedbacks in the experiment. The tasks involved in this experiment setup are detailed in the following subsections.

7.5.2.1 Constructing Recommender Peers

In this experiment, four recommenders of different organisations are constructed to simulate the proposed recommender peer interactions. These four recommenders are named as ORG1, ORG2, ORG3 and ORG4, and they are equipped with different datasets but use the same underlying recommendation technique.

By evaluating the performances of the recommenders with the same recommendation technique and different underlying datasets, we can evaluate the performance of the recommenders based on their available information sources (i.e. their underlying datasets and also collaboration from other recommender peers) without the impact from using different recommendation techniques. Moreover, the results from the experiments can also be used to verify the proposed solution to the cold-start problem (i.e. enriching the information resources from other parties).

The recommendation technique employed by the four recommenders is the standard item-based collaborative filtering technique, for detailed implementation please

refers to [3]. The use of the state-of-the-art recommendation technique ensures that our experiment can be compared and verified with other works. Moreover, it also suggested that the proposed EDRS framework and peer profiling and selection strategy can be easily adopted by existing recommenders. The main differences among the four recommenders are in their underlying datasets, specifically, they all have different customer sets (or user sets). We firstly select 6500 users from the Book-Crossing Dataset and then cluster them into 20 user clusters based on their item preferences (i.e. explicit item ratings). We denote the overall user set as U and the 20 user clusters as $uc_1, uc_2, ..., uc_20$.

From these 6500 users in U, 5000 users are selected as the training user set $\hat{U}$ (i.e. for forming the underlying datasets of the recommender peers) and the rest of 1500 users then forms the testing user set $\check{U}$, where $U = \hat{U} \cup \check{U}$ and $\hat{U} \cap \check{U} = \emptyset$. Furthermore, we denote the set of training users within cluster uc_i as $\hat{uc}_i$ and the set of testing users within cluster uc_i as $\check{uc}_i$. Importantly, the users in U are divided into the clusters first, and the 1500 users in the testing set U are then selected from each of the clusters. This process allows us to keep track of the percentages of the different user types (i.e. users in different clusters) in the testing user set.

7.5.2.2 Evaluation Metrics

The classification accuracy metrics such as Precision, Recall and F1 metrics are chosen for the performance evaluation of the recommenders against the users in the testing user set. The classification accuracy metrics are mainly based on comparing the recommended item list and the set of user preferred items. In this experiment, for each testing user $u_i \in \check{U}$, we divide the set of items explicitly rated by u_i (denoted as $\check{R}_i$) into two halves denoted by Y_i and T_i. For the two item sets Y_i and T_i, Y_i and the associated item ratings are used to represent u_i's user profile (i.e. the recommenders make recommendations to u_i based on u_i's ratings to the items in Y_i), and the items in T_i, on the other hand, are used to form the user preferred item list for evaluating the recommendations made to u_i. However, not all the items in T_i are preferred by the user u_i. The items with low rating values should not be considered as the user's preferred items because u_i has specifically indicated that they are disliked. Hence, the final testing item set $\check{T}_i$ is constructed by removing all items with ratings below u_i's average rating from T_i. For evaluating the recommenders' recommendation quality to a given testing user $u_i \in \check{U}$, the recommenders are firstly provided with u_i's profile (i.e. Y_i and the associated ratings), then the recommenders generate their recommendations to u_i, finally, the recommendations generated from the recommenders (i.e. P_i) are evaluated against the testing item set $\check{T}_i$ by utilizing the classification accuracy metrics (i.e. Precision, Recall and F1).

7.5.2.3 Benchmarks for the Peer Profiling and Selection Strategy

As mentioned earlier, one of the objectives of this experiment is to evaluate the effectiveness of the proposed peer profiling and selection technique described in Sect. 7.4. Hence, it is important to include other profiling and selection techniques as baselines in order to conclude the significance of the proposed technique. However, to the best of our knowledge, there are no other existing works available for the recommender peer profiling and selection tasks required for the proposed EDRS. As there are no existing standard baseline techniques available in distributed recommender systems, we therefore have adapt techniques from other research domains that are reasonably applicable to the required peer profiling and selection task. In this experiment, the following five peer profiling and selection strategies are compared:

- *Gittins:* the proposed recommender peer profiling and selection technique as described in Sect.7.4.
- *BPP:* Best Past Performances. It is the most fundamental and intuitive strategy being used for the profiling and selection related tasks in many research domains (e.g. the collection selection task in distributed information retrieval) . The basic idea behinds BPP is to select recommender peers with the best average past performances to the target users' belonging clusters.
- *Rand:* the manager peers based on this strategy keep no knowledge about other peers and select contractor peers at random. This strategy is included in this experiment to show the significance of having a reasonable peer profiling and selection strategy in the proposed EDRS.
- *Gittins_NC* this selection strategy is a simplified version of the proposed strategy Gittins. Essentially, Gittins_NC assumes all users belong to one cluster. Even Gittins_NC still profiles recommender peers based on their average performance, stability and selection frequency, and the selection is also based on the combined Gittins scores as described in Sect. 7.4, it does not profile the recommender peers by considering the performance differences for users in different clusters.
- *BPP_NC:* similar to Gittins_NC, this profiling and selection strategy does not differentiate peers' performance differences for users in different clusters, and it employs only the average past performances of the recommender peers to make selections (i.e. as similar to BPP). The main purpose of having Gittins_NC and BPP_NC included in this experiment is to empirically demonstrate that different recommenders have different performances towards users in different clusters.

7.5.3 Experimental Results

Each of the four standalone recommenders (i.e. ORG1, ORG2, ORG4 and ORG4) can run by itself using its own dataset. However, the performance of the individual recommenders may not be satisfactory due to the insufficiency of the dataset. The EDRS framework proposed in this chapter can improve the performances of all

involved participant recommenders by allowing them to share datasets and recommendations. Therefore, it is expected that the distributed recommendation system with a reasonable peer selection strategy outperforms the individual recommenders. Fig. 7.5, Fig. 7.6 and Fig. 7.7 present the precision, recall and F1 results obtained from running the four standalone recommenders (i.e. ORG1, ORG2, ORG4 and ORG4) and the distributed recommendation system with five peer selection strategies described in Sect. 7.5.2.3 (i.e. Rand, BPP_NC, Gittins_NC, BPP and Gittins), respectively.

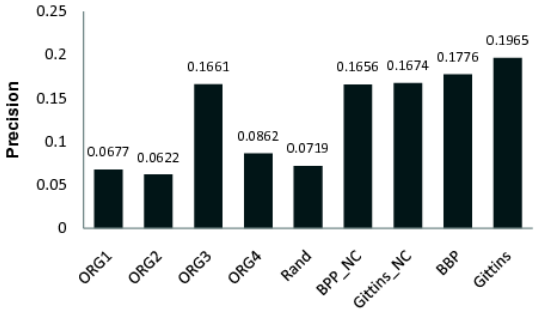

Fig. 7.5: Precision results for different recommendation settings

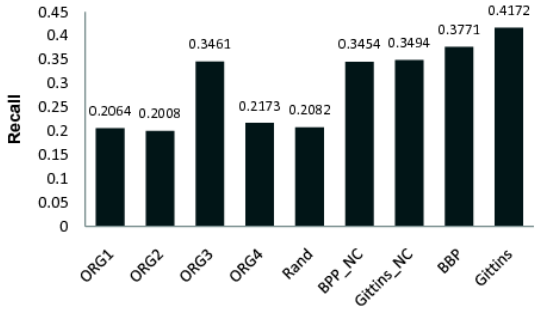

Fig. 7.6: Recall results for different recommendation settings

Let's firstly take a look at the performance of the distributed recommender system with the five different profiling and selection strategies (i.e. Rand, BPP, Gittins, BPP_NC, and Gittins_NC). Among these five strategies, Rand is the only strategy that does not have profiles for the recommender peers, and it randomly selects peers for making recommendations. Based on the experiment results shown in above figures, Rand performed the worst among all of the five strategies, and it even performed worse than two of the stand-alone recommenders ORG3 and ORG4 which make recommendations only based on their own datasets. By contrast, the

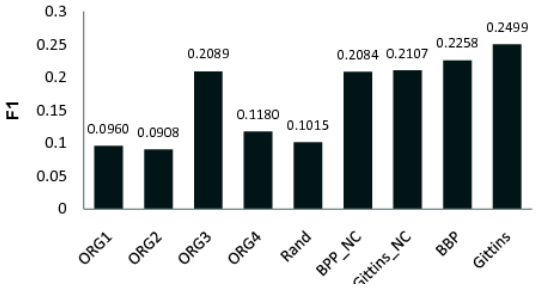

Fig. 7.7: F1 results for different recommendation settings

other four strategies (i.e. BPP_NC, Gittins_NC, BPP and Gittins) that profile recommender peers based on the peers' past performances and select peers' based on their profiles all achieved much better results than all stand-alone recommenders except for ORG3. Because ORG3 is the best performed stand-alone recommender and therefore very often selected by the manager recommender, the distributed system with some of these strategies achieved similar performance as what ORG3 does. This result suggests that by sharing datasets and selecting the most appropriate recommender to make recommendations, the distributed recommendation system can greatly improve recommendation quality. Particularly, for those peers which suffer from the cold-start problem (such as ORG1 and ORG2), the amount of improvement is significant, for instance, the performance of both ORG1 and ORG2 can be improved by more than 50% if they adapt any of the four strategies to profile and select peers.

Among the four rational strategies (i.e. BPP_NC, Gittins_NC, BPP and Gittins), BPP and Gittins profile and select peers based on their performance to users in different clusters. By contrast, BPP_NC and Gittins_NC do not consider the fact that different peers might perform differently for users in different clusters and profile peers based on their average performance over all users. As shown in 7.5, Fig. 7.6 and Fig. 7.7, the cluster based strategies BPP and Gittins significantly outperformed the noncluster based strategies BPP_NC and Gittins_NC. This is because the cluster based strategies can find the best recommender peers for making recommendations based on the target users' belonging clusters. By contrast, BPP_NC and Gittins_NC select recommender peers based on their average past performances to all users. Therefore, they will select peers performed averagely best in the past despite that these peers might be unable to produce good recommendations for some target users in certain clusters.

Finally, the experiment results show that the Gittins indices based strategies (i.e. Gittins and Gittins_NC) performed better than that of the standard performance based strategies (i.e. BPP and BPP_NC). Specifically, Gittins outperformed BPP and Gittins_NC outperformed BPP_NC. This result suggests that by combining the selection frequency and recommendation stability into peer profiling and selection

process, the best performed peers can be more accurately identified than only based on the peers' average past performances.

7.6 Conclusions

In this chapter, we suggested a new distributed system paradigm for recommenders, namely, Ecommerce-oriented Distributed Recommender System (EDRS). EDRS is designed to allow the recommenders from different organisations or parties to share datasets and recommendations with each other, so that all of them can achieve better recommendation quality and provide better services to their users. Also, as the recommenders within the proposed EDRS no longer make recommendations solely on their own efforts, they are therefore more resistant to the cold-start problems. In order to facilitate the interaction among the recommenders in the EDRS, a novel peer profiling and selection strategy is proposed in this chapter. The proposed strategy profiles and selects recommender peers based on their past recommendation performance, stability and selection frequency in cluster level, and our experiment results show that the proposed strategy allows recommender peers to effectively learn from each other and select the most appropriate peers to provide satisfactory recommendations to their users.

References

1. Baruch Awerbuch, Boaz Patt-Shamir, David Peleg, and Mark Tuttle. Improved recommendation systems. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1174–1183, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
2. Christoph Baumgarten. A probabilistic model for distributed information retrieval. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 258–266, New York, NY, USA, 1997. ACM.
3. Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.
4. Maarten Clements, Arjen P. de Vries, Johan A. Pouwelse, Jun Wang, and Marcel J. T. Reinders. Evaluation of neighbourhood selection methods in decentralized recommendation systems. In *Workshop on Large Scale Distributed Systems for Information Retrieval (LSDS-IR)*, 2007.
5. Owen de Kretser, Alistair Moffat, Tim Shimmin, and Justin Zobel. Methodologies for distributed information retrieval. In *International Conference on Distributed Computing Systems*, pages 66–73, 1998.
6. Petros Drineas, Iordanis Kerenidis, and Prabhakar Raghavan. Competitive recommendation systems. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 82–90, New York, NY, USA, 2002. ACM Press.
7. David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.
8. Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

9. Jin Li. On peer-to-peer (p2p) content delivery. *Peer-to-Peer Networking and Applications*, 1(1):45–63, March 2008.
10. Pingfeng Liu, Guihua Nie, Donglin Chen, and Zhichao Fu. The knowledge grid based intelligent electronic commerce recommender systems. In *SOCA '07: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications*, pages 223–232, Washington, DC, USA, 2007. IEEE Computer Society.
11. Brian P. McCall. Multi-armed bandit allocation indices (J. C. Gittins). 33(1), 1991.
12. Bradley N. Miller, Joseph A. Konstan, and John Riedl. Pocketlens: Toward a personal recommender system. *ACM Trans. Inf. Syst.*, 22(3):437–476, July 2004.
13. B. M. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Recommender systems for large-scale e-commerce: Scalable neighborhood formation using clustering. In *Fifth International Conference on Computer and Information Technology (ICCIT 2002)*, 2002.
14. J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, January-April 2001.
15. A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2002)*, pages 253–260, 2002.
16. Christoph Sorge. A chord-based recommender system. In *LCN '07: Proceedings of the 32nd IEEE Conference on Local Computer Networks*, pages 157–164, Washington, DC, USA, 2007. IEEE Computer Society.
17. Amund Tveit. Peer-to-peer based recommendations for mobile commerce. In *WMC '01: Proceedings of the 1st international workshop on Mobile commerce*, pages 26–29, New York, NY, USA, 2001. ACM.
18. Josã M. Vidal. *A Protocol for a Distributed Recommender System*. ACM Press, 2005.
19. Jun Wang, Arjen P. de Vries, and Marcel J. T. Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508, New York, NY, USA, 2006. ACM Press.
20. Gerhard Weiss, editor. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, Cambridge, MA, USA, 1999.