P2Prec: A Social-Based P2P Recommendation System

Fady Draidi, Esther Pacitti, Didier Parigot, Guillaume Verger

P2Prec: A Social-Based P2P Recommendation System

Fady Draidi	Esther Pacitti	Didier Parigot	Guillaume Verger
LIRMM	LIRMM	INRIA	INRIA
Montpellier, France	Montpellier, France	Sophia Antipolis, France	Montpellier, France
Fady.Draidi@lirmm.fr	Esther.Pacitti@lirmm.fr	Didier.Parigot@inria.fr	Guillaume.Verger@inria.fr

ABSTRACT

P2Prec is a social-based P2P recommendation system for large-scale content sharing that leverages content-based and social-based recommendation. The main idea is to recommend high quality documents related to query topics and contents held by *useful* friends (of friends) of the users, by exploiting friendship networks. We have implemented a prototype of P2Prec using the Shared-Data Overlay Network (SON), an open source development platform for P2P networks using web services, JXTA and OSGi. In this paper, we describe the demo of P2Prec's main services (installing P2Prec peers, initializing peers, gossiping topics of interest among friends, key-word querying for contents) using our prototype implemented as an application of SON.

Categories and Subject Descriptors

H.3.4 [**Systems and Software**]: Distributed systems, H.3.3 [**Information Search and Retrieval**]: Search process, H.3.1 [**Content Analysis and Indexing**]: Indexing methods.

General Terms

Algorithms, Experimentation

1. INTRODUCTION

The general problem we address is large-scale content sharing for on-line communities. Consider, for instance, a scientific community (e.g., in bio-informatics, physics or environmental science) where community members are willing to share large amounts of documents (including images, experimental data, etc) stored in their local servers. Assume also that they don't want to lose control over their data at a central site. A promising solution is to organize community members in a peer-to-peer (P2P) overlay network, with the advantages of decentralized control, peer autonomy and scalability.

Locating contents based on content ids in a P2P overlay network is now well solved (see e.g. [5]). However, the problem with current P2P content-sharing systems is that the users themselves, i.e., their interest or expertise in specific topics, or their rankings of documents they have read, are simply ignored. In other words, what is missing is a recommendation service that, given a query, can recommend relevant documents by exploiting user information.

Sinha et al. [10] have shown that in general users prefer the advices coming from known friends in terms of quality and trust because usually users trust their friends' advices. In most of existing P2P solutions, friendship links are extracted from user's behaviors [2], or are established based on explicit trust declaration [9]. To enrich these solutions, we exploit the fact that users who store similar contents are potential friends. Therefore, our solution leverages between content-based and social-based recommendations over a P2P overlay.

P2Prec[1] is a social-based P2P recommendation system for large-scale content sharing [4]. The main idea is to recommend high quality documents related to query topics and contents hold by *useful* friends (of friends) of the users, by exploiting friendship networks. Our recommendation model relies on a distributed graph, where each node represents a user (peer) labelled with the contents it stores and its topics of interests. The topics each peer is interested in are automatically calculated by analyzing the documents the peer holds. Peers become *relevant* for a topic if they hold a certain number of highly rated documents on this topic. A peer *v* becomes useful to a peer *u*, if *u*'s topics of interest and *v*'s relevant topics are overlapped. To exploit friendship links, we rely on Friend-Of-A-Friend (FOAF) descriptions (http://www.foaf-project.org). To disseminate information about relevant peers, we rely on gossip algorithms that provide scalability, robustness, simplicity and load balancing. In addition, we propose an efficient query routing algorithm that selects the best peers to recommend documents based on users' useful friends and query topics. At the query's initiator, recommendations are selectively chosen based on similarity, rates and popularity or other recommendation criteria.

We have implemented a prototype of P2Prec by using the Shared-Data Overlay Network (SON) (http://www-sop.inria.fr/teams/zenith/SON), an open source development platform for P2P networks using web services, JXTA and OSGi (http://www.osgi.org). SON components communicate by asynchronous message passing to provide weak coupling between system entities. To scale up and ease deployment, we rely on a Distributed Hash Table (DHT) for publishing and discovering services or data.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK,

[1] Work partially funded by the Dataring project of the Agence Nationale de la Recherche.

In this paper, we describe the demo of P2Prec's main services (installing and initializing P2Prec peers, gossiping topics of interest among friends, key-word querying for contents) using our prototype implemented as an application of SON.

2. OVERVIEW OF P2Prec

Centralized systems recommender systems (RS) rely on the ratings that users provide [1]. The advents of Web2.0 tools and the growing popularity of online social networks have led to the development of social-based RS that use users' social data such as friends, trust, etc. to provide recommendations [9]. These systems exploit the preferences and relations of users' friends (of friends) [2] or the trust relations [8] between users to aggregate the neighbors of each user. Then the recommendations are computed based on the ratings that have been given by those neighbors.

P2Prec's recommendation model is expressed based on a graph $G = (D,U,E,T)$, where D is the set of shared documents, U is the set of users $u_1,...u_n$ corresponding to autonomous peers $p_1,...p_n$, E is the set of edges between the users such that there is an edge $e(u,v)$ if users u and v are friends, and T is the domain of topics. Each user $u \in U$ is associated with a set of topics of interest $T_u \subset T$, and a set of relevant topics $T_u^r \subset T_u$ extracted locally from the documents u has rated. The rating that has been given by a user u on document doc is denoted by $rate_{doc}^{u}$.

In our approach, we use Latent Dirichlet Allocation (LDA) [3] to automatically model the topics in the system, which in turn are used to extract users' relevant topics of interest. LDA processing is done in two steps: training at a global level and inference at the local level. The global level is given to a bootstrap server (BS), where BS aggregates a sample set of M documents from P2Prec participant peers. Then BS executes the LDA classifier program to get a set $T = \{t_1,..t_k\}$ of topics, where k is the number of topics. Each topic $t \in T$ contains a set of z words, where z is the total number of the unique words in M, and each of these words is associated with a weight value between 0 and 1. At the local level, user u performs LDA locally to extract the topics of its local documents, using the same set of topics T that were previously generated at the global level. LDA provides a vector of size k for each document doc, $V_{doc} = [w_{doc}^{t1},...,w_{doc}^{tk}]$, where w_{doc}^{t} is the weight of each topic $t \in T$ with respect to doc.

Users' relevant topics of interest are extracted based on a combination between documents' semantics and ratings. Once a user u extracted the V_{doc} for each $doc \in D_u$, it multiplies the $V_{doc} = [w_{doc}^{t1},...,w_{doc}^{tk}]$, by the rating $rate_{doc}^{u}$. Then, user u identifies for each topic $t \in T$ only the documents that are highly related to t. A document doc is considered highly related to topic t, if its weight in that topic w_{doc}^{t} multiplied by its rating $rate_{doc}^{u}$ exceeds a threshold value. Next, u counts how many documents are highly related to each topic $t \in T$. User u is considered interested in topic $t \in T_u$ if a percentage y of its local documents are highly related to topic t. Finally, u is considered a *relevant user* in topic $t \in T_u^r$ if it is interested in t and has a sufficient amount of documents that are highly related to topic t.

Each user $u \in U$ maintains a FOAF file that contains a description of its personal information, and friendship network. Personal information includes the extracted topics of interest, where each topic of interest $t \in T_u$ is associated with a Boolean value which indicates whether u is relevant in that topic. Friends information includes friends' name, links (URI) to their FOAF files, relevant (topics of interest), and trust levels. The trust level between user u and a friend v, denoted by $trust(u,v)$, is a real value within [0, 1], and it represents how much user u has faith in its friend v.

Furthermore, each user u establishes new friendships with users that are *useful* to u's queries, and if their friendship networks have high overlap with u's friendship network. A user v is considered *useful* to a user u, if v is a relevant user and a certain amount of v's relevant topics T_v^r are of interest for u. user u keeps locally (in its FOAF file) its useful friends and their corresponding relevant (topics of interests). User u exploits its useful friends (of friends) for recommendations.

To establish friendship and disseminate recommendation, we rely on gossip protocols [7] as follows. At each gossip exchange, each user u checks its gossip *local-view* to enquire whether there is any relevant user v that is useful to u, and its friendship networks have high overlap with u's friendship network. If it is the case, a demand of friendship is launched among u and v and the respective FOAF files are updated accordingly. FOAF files are used to support users' queries. Whenever a user submits a key-word query, the FOAF file is used as a directory to redirect the query to the top-k most adequate friends by taking into account similarities, relevance, usefulness and trust.

Finally, a key-word query q is associated with a TTL (Time To Live) and is routed recursively in a P2P top-k manner: once a query is submitted by u, it is forwarded to u's top-k useful and trustful friends. When a query is received at any peer, it is again redirected to its top-k useful and trustful friends, until TTL is reached. Each user v that received the query provides recommendations to u. The response to a query q is a recommendation that has been provided in a ranked list, based on a function that ranks each document according to its relevance with q, its popularity, the similarity and trust between q's initiator and responder v.

3. P2Prec IMPLEMENTATION

With SON, the development of a P2P application is done through the design and implementation of a set of components. Each component includes a non-functional code that provides the component services and a code component that provides the component logic (business code). The complex aspects of asynchronous distributed programming (non-functional code) are separated from code components and automatically generated. From a description of a component's services, the Component Generator (CG) automatically generates the non-functional code. Thus, the programmer does not deal with complex distributed programming aspects.

The basic infrastructure of SON is composed of a Component Manager (CM), a Publishing and Discovery Component (PDC), and a Connection Component (CC). The PDC allows publishing or discovering components on different peers using a DHT. The CC provides connection between remote components on peers. The CM performs the creation of new component instances and the connections between them.

To establish a connection between two components, the CM uses the services description of each component. At run-time, when a component A wants to connect with another component B, it must use the service `ConnectTo(A,B)` provided by the CM. As in the CM that created the component, the components are by default connected to the CM. To establish a connection between two components, the CM uses the services description to associate the

services provided by component A with the services required by component B, and conversely.

After the connection process, the two components can communicate directly with each other without going through the CM. The advantage of this process is that it is done at run-time, thus avoiding each component to know statically the services of other components. In fact, each component can, on the fly, connect to any component. The assembly of components of a given application is not necessarily known statically and can evolve dynamically over time. Thus, the components are autonomous and independent.

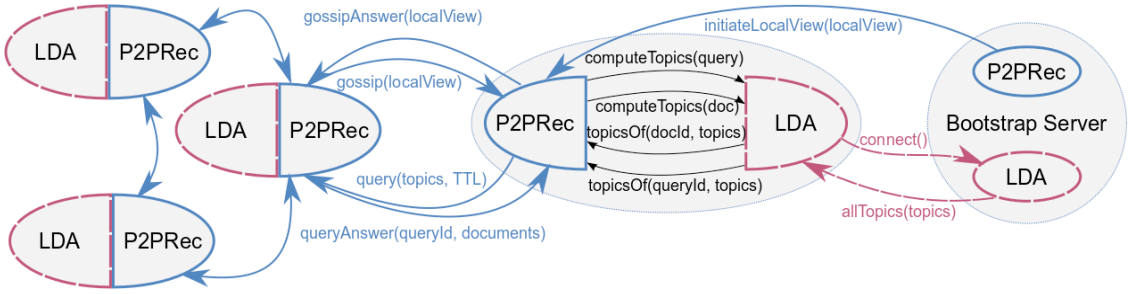

Figure 1: P2Prec Architecture.

The CM delegates the management of lists of remote components to the PDC. In the current version, we use the OpenChord DHT implementation (http://open-chord.sourceforge.net/) for the PDC, but nothing prevents us from using other implementations. For this purpose, an interface has been defined with the usual methods (`put (key,value)` and `get(key)`) that can be expected from a DHT. At each creation of a component, the CM publishes into this DHT the information for a remote component useful to connect to this component.

The Connection Component (CC) is a component that handles the communication between remote components. It opens the TCP connection between peers. It is based on the concept of virtual pipes introduced by JXTA technology (http://jxta.kenai.com/). This concept allows passing through a single TCP connection several logical communications (virtual pipes) between peers. Using this abstraction allows each component to open a virtual pipe to read messages sent to it. We identify a virtual pipe by a universally unique identifier (UUID).

SON is implemented in Java on top of OSGi components that provide all basic services for the lifecycle of our components, in particular, the deployment services. The launching of a SON application is defined through an OSGi configuration, which describes the application components.

3.1 P2Prec ARCHITECTURE

We developed P2Prec as a SON application (see http://www-sop.inria.fr/teams/zenith/p2prec/) with two components: the LDA component for the documents topics process and the P2Prec component for the recommendation process. For instance, the services of the P2Prec component are the services for passive and active propagation through gossip services (*gossip* and *gossipAnswer* services) and the queries services (*query* and *queryAnswer* services). There are two OSGi configurations, the Bootstrap Server (BS) configuration and the Client (the peer) configuration, as shown in Figure 1.

To run the P2Prec application, the BS must be started on a given machine (with a given IP address). This IP address will be used as the entry point into the P2Prec network for new peers. At the startup time, a new peer must first identify itself with the BS (*connect* service) and the BS is going to return the current set of all topics (*allTopics* service). Then within the local peer's LDA component and the current topics, the topics of each document is computed locally.

After these steps, the peer can start the recommendation steps and documents discovery without any connection with the BS. Indeed, the research of topics of a new document (*computeTopic(doc)* service) and the computing of topics of a query (*computeTopic(query)* service) can be made locally with the local peer's LDA component. Depending on the evolution of documents on the P2Prec network, the BS may update the set of topics of documents, and inform the peers by broadcasting this new topic set (using the *allTopics* service).

4. P2Prec DEMONSTRATION

In this section we describe how the P2Prec services cooperate using scenarios based on the Ohsumed documents corpus [6] that has been widely used in IR. It is a set of 348566 references from MEDLINE, the on-line medical information database, consisting of titles or abstracts from 270 medical journals over a five year period (1987-1991). Our application is lightweight, meaning that no client needs to be downloaded to use it.

In order to couple the P2Prec core, made of OSGi configurations, with the chosen scenarios, we used the Google Web Toolkit (GWT: http://code.google.com/webtoolkit/) to build the user interface. This toolkit allows defining a client/server application written completely in Java that runs in a web browser. It automatically compiles the Java client code into HTML and JavaScript, and easily permits to use Java libraries. Therefore, all graphical interfaces of this demo are made of web pages and run in a classical browser. We show how the application works, from the global installation to the utilization by an end-user.

Installation In order to run a P2Prec peer properly, any user (at a peer) has to connect first to the Bootstrap Server (BS). Therefore we define a place the BS will run on. Every peer in the system will know its IP address. As the BS and any peer offer the same kind of services, we have defined two OSGi configurations for running P2Prec components: one as a BS, and one as a standard peer that will connect to the BS.

Initialization Each peer consists of a LDA part coupled with a Communication part (called P2Prec). As the demonstration starts, the BS is created, and so are several peers (30 of them). Each peer sends some of its documents, which are arbitrarily distributed

among all peers, to the BS to perform LDA on a sample of all documents and to define the set of topics used in the network. Next, the BS informs all connected peers about the topics that are present in the network, and each peer indexes its own documents with the set of topics. Each peer is given an initial FOAF, which determines its friends in the network, and provides it information about them. It can now start gossiping with other peers, and the user belonging to the peer can send queries to discover documents. When connecting a new peer to the network, we show how it gets initial information in its FOAF file in two cases: (1) it has already joined the network in the past (i.e. it knows other peers); (2) it connects to the network for the first time.

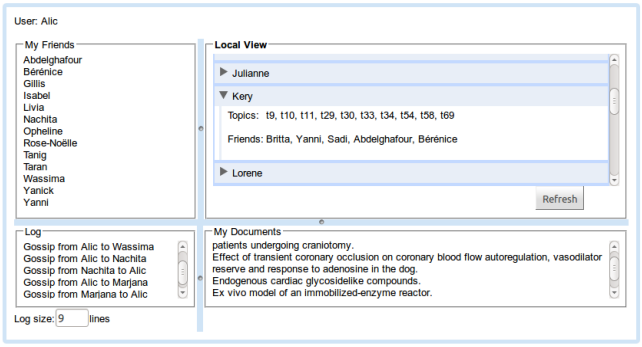

Figure 2: P2Prec Gossip Interface.

Gossiping The gossip service is at the heart of P2Prec, and is transparent to the end-user. While peers exchange gossiping messages, the system recommends new friendships to users. For the sake of the demonstration, we developed an interface showing what is internally happening during gossiping (see Figure 2). The interface shows the current friends of the user, the gossiping messages sent and received by the peer, the gossip local-view that permits to find friends, etc. We show how the gossip mechanism notifies the user that other users share the same interests, and ask her to add them to her friend list.

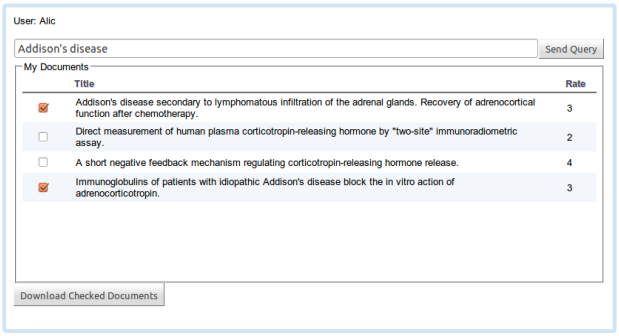

Figure 3: P2Prec Query Interface.

Querying Spreading information with gossip to make new friends has one aim: being able to answer queries accurately when a user searches for documents. This is where the query service is needed. The user is able to send a query for getting documents recommendations from her friends. The local LDA of the user translates the query into a set of relevant topics, and the peer sends them through the query service to the user's friends. Each friend may recommend documents depending on the similarity in terms of topics and the rate of the document. The query hops to friends of friends as many times as its TTL allows, the results being returned during the journey.

Figure 3 shows the result returned to the user after a query is sent. We show the results of the query for a user who has been in the network for a long time compared to a new user, and compare the accuracy and the number of answers she gets.

The demo with the MEDLINE information database (with a fixed data set) can be downloaded from the P2Prec website (http://www-sop.inria.fr/teams/zenith/p2prec/) with the complete procedure for installing, deploying and running, using OSGi configurations. We also provide a second procedure for building P2Prec applications with any data set, from scratch. But before to be used by end-users, it requires setting up a Bootstrap Server and compute (initialize) the set of topics of the given data set.

5. REFERENCES

[1] Adomavicius, G., Tuzhilin, A., Towards the next generation of recommender systems: a survey of the state-of-theart and possible extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6):734–749, 2005.

[2] Arazy, O., Kumar, N., Shapira, B., Improving social recommender systems. Journal of IT Professional, 11(4):38–44, 2009.

[3] Blei, D. M., Ng, A. Y., Jordan, M. I., Latent Dirichlet Allocation. Journal of Machine Learning, 3:993–1022, 2003.

[4] Draidi F., Pacitti E., Kemme B. P2Prec: a P2P recommendation system for large-scale data sharing. Journal of Transactions on Large-Scale Data and Knowledge-Centered Systems (TLDKS), Vol. 3, Springer, LNCS 6790, 87-116, 2011.

[5] El Dick M., Pacitti E., Akbarinia R., Kemme, B. Building a peer-to-peer content distribution network with high performance, scalability and robustness. Information Systems, 36(2):222-247, 2011.

[6] Hersh W.R., Buckley C., Leone T., Hickam D.H., Ohsumed: An interactive retrieval evaluation and new large test collection for research. ACM SIGIR, 192-201,1994.

[7] Jelasity M., Voulgaris S., Guerraoui R., Kermarrec A.M., VanSteen M. Gossip-based peer sampling. ACM Trans. On Computer Systems, 25(3):2007.

[8] Kim, H.-J., Jung, J.J., Jo, G.-S.: Conceptual framework for recommendation system based on distributed user ratings. Springer, LNCS 3032, 115-122, 2003.

[9] Massa, P., and Avesani P. 2004. Trust-aware Collaborative Filtering for Recommender Systems. Springer, LNCS 3290, pp. 492-508, 2004.

[10] Sinha, R., Swearingen, K. Comparing Recommendation made by Online Systems and Friends. DELOS-NSF Workshop on Personalization and Recommender Systems in Digital Libraries, 2001