Personalization as a Service: the Architecture and a Case Study

Hang Guo, Jidong Chen
EMC Research China, Beijing, China
{guo_hang, chen_jidong}@emc.com

Wentao Wu, Wei Wang
Fudan University, Shanghai, China
{wentaowu,weiwang1}@fudan.edu.cn

ABSTRACT

Cloud computing has become a hot topic in the IT industry. Great efforts have been made to establish cloud computing platforms for enterprise users, mostly small businesses. However, there are few researches about the impact of cloud computing over individual users. In this paper we focus on how to provide personalized services for individual users in the cloud environment. We argue that a personalized cloud service shall compose of two parts. The client side program records user activities on personal devices such as PC. Besides that, the user model is also computed on the client side to avoid server overhead. The cloud side program fetches the user model periodically and adjusts its results accordingly. We build a personalized cloud data search engine prototype to prove our idea.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

General Terms

Design, Human Factors

Keywords

Cloud Computing, Personalization as a Service

1. INTRODUCTION

Cloud computing has the potential to change the IT industry. The hardware and software resources in the cloud data centers can be packaged as services. Business runners can purchase these services with considerably lower prices (taking the maintenance cost into consideration) and manage the services with great flexibility. Business runners do not have to make a plan far ahead. They can start with less computing services and purchase more when needed. Indeed, enterprise users, especially small business runners, will benefit from cloud computing. On the other hand, it is still unclear that whether the notion of cloud computing will greatly influence the lives of individual users in the future.

Individual users will create most of the digital universe. A recent report from IDC[1] shows that "nearly 70% of the digital universe will be generated by individuals by 2010". However, individual users are not capable of managing their own data. The same report claims that "organizations will be responsible for the security, privacy, reliability and compliance of at least 85% of the information." Cloud computing will impose great impact on the IT industry. And it will affect the user experiences of individual users as well.

Nowadays, there are many applications and services for individual users. Some are bounded with certain platforms such as PCs and mobile phones. Some are web-based services such as email, blog. According to Berkeley's vision of cloud computing [1], web-based services will be the mainstream at the age of cloud. Currently only large service providers i.e., Google, Amazon, can afford to build up-front data centers and maintain their services with high quality. Cloud computing enables small groups to step in the end user market. Small companies can also set up highly available services. As a result, individual users will have much more choices than they have today. In other words, the competition in this area will be fierce.

Individual users have some important concerns when they are choosing between different service providers. Among them, personalization is critical. The demand for personalized services increases as information explodes on the cloud. A study from IDC [2] shows that "not enough ability to customize" is among the top 5 challenges ascribed to the cloud service model. Nowadays the most popular practice is to customize services with user profiles. Users have to choose their interests from predefined templates such as "Sports", "Games", etc. This approach is adopted by many large services providers but it will face many challenges in the cloud age.

- Every service provider has its own user repository. It is extremely hard, if possible, to integrate one repository with others. As a result, users have to maintain their information for all the services they used.
- Users will be too tired to edit profiles for all kinds of cloud services. It will be better if the model is build automatically.

[1]http://alexbarnett.net/blog/archive/2007/03/06/The-Expanding-Digital-Universe.aspx
[2]http://blogs.idc.com/ie/?p=210

CloudDB'09, November 2, 2009, Hong Kong, China.

- Users prefer unique experiences. However, sometimes they are only treated as groups.
- User information gathering is the key to build user models. Cloud-side activity recorder only works when the user is on-line. However, users will spend most of their time offline or working with other cloud services.
- The service provider will be overcharged if the user repository is maintained on the cloud.

These challenges will bring great opportunities to develop new personalization solutions for cloud service providers. In this paper we first analyze the above challenges. We argue that the new personalized solution should be built on the client side instead of the cloud side.

The personalized module consists of three parts: the user information collector, the user model builder and the model synchronizer. The user information collector gathers all the user activities on the user's PC. The model builder computes/updates the user model periodically according to the recent user activities. When a cloud service needs the user's data at a specific moment, say 2 days ago, the model synchronizer will upload the corresponding users model to the cloud. We call this personalization paradigm Personalization as a Service (PaaS).

In this way the personalization module is independent from various types of cloud services. Its major task is long-term user information gathering and modeling. The user model is shared by all cloud services. So the user is able to enjoy the same experience in different cloud services. Moreover, the user model is built on his own log. Therefore every user has unique user experience. Another advantage of this paradigm is performance. The server in the cloud is not overloaded with the model construction/update costs of millions of users. The model is built and updated at the client side.

We implement this architecture with a client-side personalization module and a cloud-side user data search module. The cloud-side application backups use data at regular intervals. Users can search their data with the built-in full-text search engine. With the help of the personalization module, the search result of every user is personalized. The documents are re-ranked by their relevances to the user's current interests.

The paper is organized as following. In the second section we analyze the challenges of personalized services in the cloud age. Section 3 introduces our PaaS architecture. Section 4 gives a case study on personalized search. The related work is shown in Section 5. Finally the paper is concluded in Section 6.

2. CHALLENGES

Personalization is essential for individual users. As mentioned in the first section, cloud computing brings new challenges on personalization to the cloud service providers. In this section we go through all these challenges and analyze their potential impacts.

2.1 Shared Model VS Private User Model

Today most service providers have their private user repository which includes all users' information. The repository is composed by user models. The user model may be a user's profile based on predefined template, or a complicated

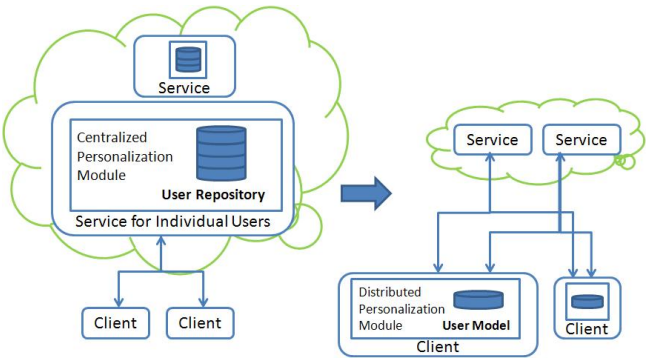


Figure 1: Shared User Models VS Private User models

mathematical model. No matter what kind of model is employed, the user repository is regarded as a critical asset of the company and never shared with others for business purpose. Therefore a user have to register all the services separately. Moreover, when the user's preference changes, he must update his information in all the repositories.

The second problem is that the structure of the user repository is rigid. If the service provider wants to know more about his customers to support new services, there are two solutions. First, the provider collects more user activities to build more powerful models. Second, all users must go back to the registration page and update their profiles. Both solutions take a long time.

The third problem comes from small business owners. Cloud computing makes it much easier for small business owners to set up stable web services for individual users. However, it is not easy to customize their services from scratch. It may take years to collect enough user information to build a user repository. Moreover, the service provider has to update the user repository periodically to keep it up-to-date. It may be a great burden as the number of users increases.

Our idea is to decentralize the user repository and push all user models down to client side, as shown in Fig 1. It is the user himself who deals with his own preference. The user model is built and maintained by the client-side application. When the user is interested in some cloud services, he can upload his model so the service provider can pick up some of his preferences and put that into its user repository on the cloud. When the user's interests change, he only needs to update the local user model and synchronize it with all registered services. The service provider will have more flexibility to choose different aspects from the user models to facilitate new services. Start-ups and small companies can also benefit from this idea. They can have detailed user information in the first day. And users will maintain their information themselves.

2.2 Ubiquitous Information Collection VS Service-Specific Information Collection

Current service providers rely on built-in personalization modules to customize services for different users. The personalization module usually includes a backend information collector and a user model constructor. Personal information is the key to build user models. Service providers make

a lot of efforts to collect user information. As soon as the user signed in, the information collector begins to gather his information. With more user information, there is better chance to estimate the user preferences correctly. In most cases, when the user signed out, the information collector cannot get his information any more. If the user frequently switches between services, each service can only get limited information about the user. If the user is working offline, his information is not available for any service.

An intuitive idea is to run the use information collector at the client side. This collector is independent with any services. Its only responsibility is to gather all the user information no matter whether the user is online and which service is being used. This information is used to build a user model that describes the user's interests in general.

An important concern about ubiquitous information collection is the user's privacy. Indeed service providers know more about the user himself by employing the ubiquitous information collection approach. However, they do not have access the original user information such as browsing logs, contents of documents read/written by the user. These original information are analyzed at the client side. Service providers only have the user model which describes the user's preference. In this way the user's privacy is protected as much as possible.

2.3 Explicit Information Collection VS Implicit Information Collection

Another question is what kind of information should be collected at the client side. Many web portals explicitly ask their users to provide their personal information when subscribing their services. Use profiles are built accordingly. Then subscribers can get personalized services based on their profiles. A good example of such portals is MyYahoo! [3].

Another explicit way of collecting user information is to ask them for feedbacks. A lot of web service providers design fancy interfaces to encourage users to rate their objects, e.g. web pages, videos, product descriptions. These feedbacks are often used to train a user model which estimates the user's personal interests. The problem is that it takes many feedbacks from a single user to train a well-functioning user model.

Both of these explicit methods place additional burden on the user. And because the user model is not shared by service providers, the user has do it all over again when he subscribes a new service. It can be a problem when new service providers emerges on the cloud. Some service providers may not get enough user information if they employs the explicit information collection methods described above.

Implicit user information collection methods, which do not require any explicit human intervention, may be a solution to the problem. There are already many researches on implicit user feedbacks [4] such as browsing history, click-through data, search history, etc. The implicit method is an important supplementary to the explicit method. It helps to get more information from the lazy users, which can be used to train better user models.

A client-side user information collector works as following. The user can configure his profile by selecting his areas of interests, which is used to initialize the user model. The implicit user activity collector gathers all the user activities, including file access log, email log, web browsing log, etc. These activities are used to further optimize the model. By observing implicit user feedbacks for a long time, the collector may have enough user information to build a model that is able to predict the user's behavior. Service providers can give make better personalized services with the model.

[3]http://my.yahoo.com

2.4 Distributed Modeling VS Centralized Modeling

Building a user model is a time consuming task. It costs a lot of CPU time to build thousands of user models, let alone update these models periodically. Cloud infrastructure providers, such as Amazon, charge by the CPU time and the I/O bandwidth. Therefore it is expensive to build and update all user models on the cloud side.

By building and updating the user model on the client side, the service provider is able to save most of the cost. There may be extra I/O when uploading the user model. However, most user models takes very little spaces. In our case, the user model takes only a few kilobytes. The extra cost is trivial comparing to the CPU time cost on the cloud.

2.5 Unique Experience VS Group Experience

As for personalized services, a user usually prefers to be treated as an unique individual rather than one from a specific interest group. An ideal personalized system shall not provide two users with the same content. However, many service providers, such as MyYahoo!, only serve groups of users. There are two main reasons. First, the number of users is too large so the service provider cannot afford to give unique user experience to every user. Second, there is not enough user information for every single user.

As mentioned above, PaaS may solve these two problems. First of all, the user model is computed on the client side. Most of the CPU time is saved. Second, a client-side information collector can produce enough user information for one user. By carefully analyzing the information, it is possible for every user to have unique experience.

In general, PaaS has following advantages comparing with traditional server-side personalization methods in the coming cloud age. First, it makes it possible for different service providers to share an existing user model. Second, the client side user information collector produces more information than cloud side collectors. It is possible to build better user models. Third, the information collector makes full use of the implicit user feedbacks in the client side. It saves a lot of human efforts. Fourth, PaaS helps the service provider save a lot of expenses of building and maintaining the user model. Fifth, PaaS brings unique user experience to every single user.

3. THE ARCHITECTURE

In this section we introduce our PaaS architecture. Fig 2 gives a basic picture of PaaS. The dashed rectangles/polygons are the main components of PaaS.

The user model is computed at regular intervals, which are called *Update Cycles*. In each cycle, the user's information are captured by the user information collector. Based on the information, the model builder constructs the user model, which is stored in the user model store. If the user wants to share his model with trusted service providers, the model synchronizer uploads selected models to the service providers. In this way the providers know better about that

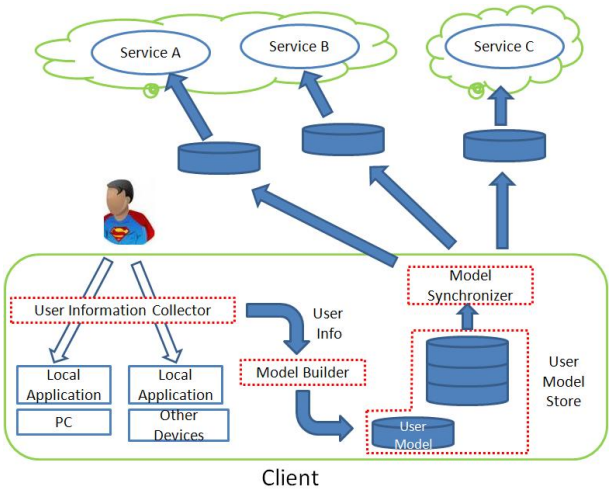


Figure 2: PaaS Architecture

user and they are able to provider personalized service with relatively lower cost.

3.1 User Information Collector

The user information collector gathers all the information of the user, which may include

- interested topics. Users may select some topics from predefined templates.
- explicit feedbacks. Explicit feedbacks refer to the explicit interventions of the user. Tags that manually assigned to documents by the user are such examples.
- implicit feedbacks. Implicit feedbacks are mostly user logs such as file access log, browsing log, email log, message log, etc. Plug-ins may be required to get some application-specific activities.
- documents read by the user. These documents include office documents, pdf files, emails, received instance messages, web pages, etc. By analyzing the contents of these documents, the user model may estimate the user's special interests.
- documents written by the user. These documents include office documents, emails, instance messages send by the user, etc. These documents are more important than others because the user's ideas are in these documents.

3.2 Model Builder

Considering the fact that the user's interests changes over time, the model builder computes a user model in every update cycle. A model only manifests the user's interests in the corresponding update cycle. Every user model has a timestamp when it is created, which is used as the index of user models in the user model store.

With plenty of user information, it is possible to build complex user models. Some mathematical models, such as Hidden Markov Model [7] (HMM), are very effective in estimating user's behavior. Service providers cannot afford to build such complex model on the cloud.

It is also possible to use different models to analyze different activities. Some models [10] are effective in processing clickthrough data. Other models like SVM [9] are good at analyzing the contents of documents. By combining these models, the service provider can have a better understanding of the user's preferences. These models take much less spaces than original user logs, therefore it is possible to build a store of user models on the client side.

3.3 User Model Store

The user model store is a repository of user models of different update cycles. In our case study, we use MySQL [4] as the user model store. It is also possible to employ key-value stores such as Berkeley DB [5].

The indexes are very important for the user model store. As mentioned in Section 3.2, timestamp is used as the index of user models. However, sometimes there is no clear clue about the exact time. Other indexes could be documents that are frequently accessed in every update cycle, important keywords read/written by the user, special events happened at that time, etc.

3.4 Model Synchronizer

The model synchronizer sends user models to the services on the cloud. Its main responsibilities are:

- privacy control. This module is configured by the user himself. If the user wants to share his model with a trusted service provider, he puts it into the white list of the synchronizer. The user can share part of the model with a service provider. The black list is used to block unwelcome services. For instance, if the user doesn't want anyone know the contents of his documents, the corresponding model such as SVM will not be shared.
- user model retrieval. The model synchronizer can use standard query language to retrieve models from the user model store. For example, if the store is a relational database system, it uses SQL.
- user model dispatching. The synchronizer handles all the connections with remote servers. It should support multiple protocols such as TCP/IP, sockets, web services, etc.

4. A CASE STUDY

In this section we give a example of PaaS, our personalized cloud search prototype. Today personal data has become one of the most important personal assets. Many companies are providing their on-line backup services. Our prototype backups user's desktop resources, i.e. local files, emails, browsed web pages, on the cloud and builds full text indexes for these resources. Different from other on-line backup services, our prototype also provides personalized search service.

Our service employs a HMM model to estimate the user's behavior. According to our model, the user has several tasks to do in each update cycle. Every desktop resource is related to one or more tasks. The user jumps from one task to another when he is opening desktop resources. A example of

[4]http://www.mysql.com
[5]http://www.oracle.com/technology/products/berkeley-db/

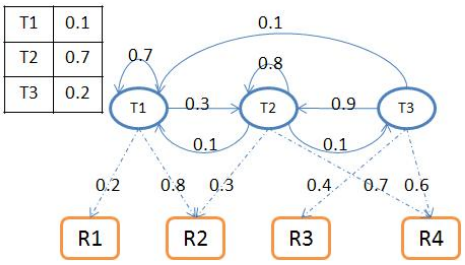


Figure 3: An Example of HMM

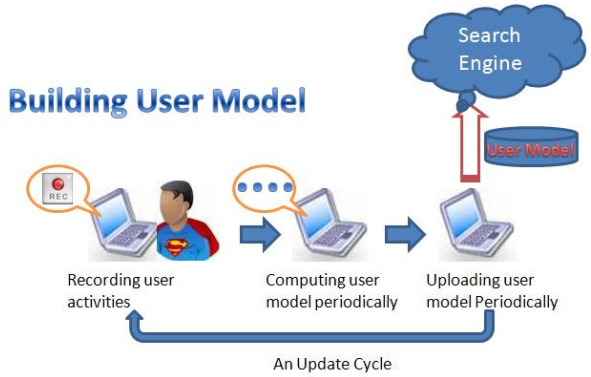


Figure 4: How to Build the Model

HMM is shown in Fig 3. The user starts to work on one task (T1 or T2 or T3) according to the predefined probabilities (the left table in Fig 3). As a result, he accesses a resource (R1 or R2 or R3 or R4) which is related to that task. The numbers on the dashed lines show the probabilities that the resources are related to the tasks. After that the user transforms to another task. The numbers on the real lines show the probabilities of the transformation from one task to another.

Our prototype works as following. First the user's desktop resources are uploaded to the cloud. Then the cloud side search engine build full text indexes on the cloud. When the user is using his PC, the information collector gathers the user information on the client side. At the end of an update cycle the model builder computes the user model. The detail is not introduced in this paper. When the new model is finished, it is uploaded to the cloud.

When the user submits a query to the remote search engine, the client estimates his current task according to our user model and then submit the query as well as his current task to the cloud. Then the search engine returns a rank list which is a combination of the resource's similarity score and relevance score. Suppose S is the score of a document in the result list,

$$S = \alpha * Relevance\_Score + (1 - \alpha) * Similarity\_Score$$

The similarity score shows how this resource is related to the query. And the relevance score measures how this resource is related to the user's current task. In this way the user can have the resource which is most relevant to his current statues.

This personalized search method has been integrated into iMecho, an associative memory based desktop search system [2]. Figure 6 shows the user interface of the iMecho prototype system.

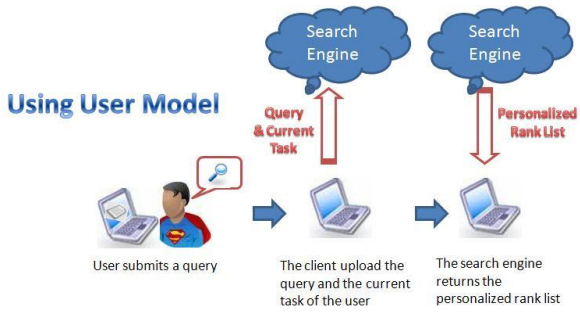


Figure 5: How to Use the Model

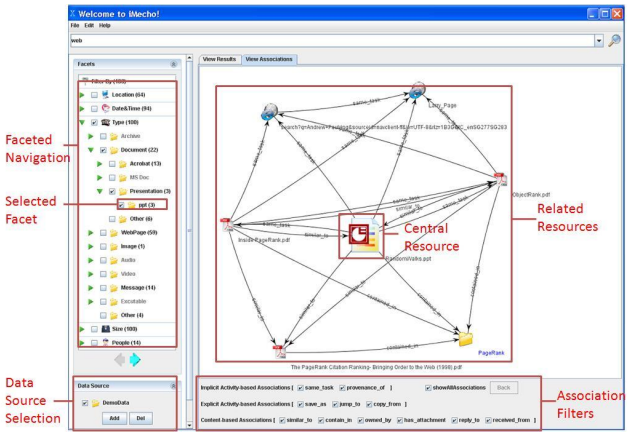


Figure 6: System User Interface

4.1 User Study Design

Five of our colleagues are invited to take part in the experiment, which is divided into two sessions. The first session is the training stage which lasts a whole day. Our iMecho prototype are installed on each PC of these five users. We briefly introduce the main idea of our system, especially the concepts of context and tasks, and give necessary training to help them with iMecho. Next, we ask them to upload a folder that contains all desktop resources they will open in both sessions. Then each of them uses iMecho to build indexes on the selected folder. In this session, the participants just perform their daily tasks and iMecho's event monitor records the user's desktop activity.

The second session is conducted in the following day.Before the session begins, the activity log produced by the event monitor in the first session are used to train the user model, which will be used in this session. They are also told to work as yesterday and in the same folder. When using iMecho to search for their resources, each participant is given a survey form to fill each query submitted in the session, together with a short description of what he is really doing when submitting the query. Our ranking results are compared with the results of Lucene, a popular open source full-text search engine. In our experiment, Lucene only ranks results by similarity scores. The participants should decide which one is better. A simple yes-or-no relevance standard is used. As

for the personalized ranking method, the participants should set the tuning factor α to 0.2, 0.5 and 0.8 for every query. Furthermore, We encourage the participants to submit the same query when they perform different tasks.

Experiments were run on a Intel PC with 2GB of memory, running the Window Vista and Sun's Java 1.6.0 JVM. The average data set contains 9,431 desktop files in 1,019 directories. The average directory depth is 9 with the longest being 15. On average, directories contain 10.3 sub-directories and files, with the largest containing 241. 75% of the files are smaller than 16 KB, and 95% of the files are smaller than 40 KB. The largest file is of size 21.5MB. The user log produced by the event monitor records totally 1,601 desktop events.

4.2 Evaluation of User Model

We conduct our model construction algorithms on the dataset described above. All 5 tasks from user A is shown in Table 1. To save space, we only list the relative path of the resources.

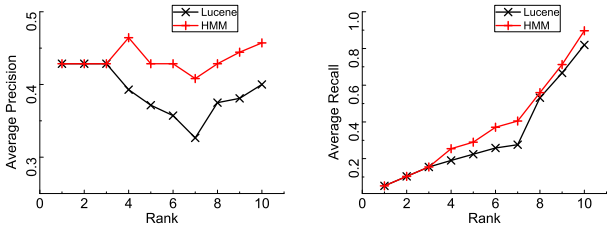
A quick look at the file paths and names in each task gives us an impression that the resources within the same task are semantically relevant. We further interviewed with user A and confirmed that these tasks indeed reflected his behavior in the first session. Since user A was recently engaged in a project on desktop search engines, the majority of his activities was concentrated on this topic, as shown in the 5 tasks. Specifically, in T1, *Lucene* is a full-text search engine, and *Sesame* is a well know RDF repository incorporating efficient RDF storage and query processing functionalities. *LuceneSail* is simply a combination of these two technologies. These resources are all related to that task. Other discovered tasks also make sense. T2 is the writing of a survey of popular ranking scheme widely used by current web search engines; T3 and T5 are the writing of two reports about *Beagle*, which is a desktop search engine on the Linux platform. They focus on Beagle's indexing and ranking methods respectively. Finally, T4 is an investigation of different desktop search tools. These results obviously prove the effectiveness of the task mining strategy we proposed during the model construction process.

4.3 Evaluation of Personalized Ranking

We next evaluate the effectiveness of our ranking scheme by comparing with similarity-based ranking scheme used in Lucene. We also study the effectiveness of our the personalized ranking approach. This is done by analyzing the relevance of the returned results to the keyword queries submitted by the participants during the second session. In average, 5 queries are issued (Table 2 lists the queries of user A) and the average query length is 2.2 keywords.

Table 2: Queries issued by user A in the second session

Query	Length
beagle indexing	2
beagle ranking	2
desktop search application	3
PageRank model	2
random walk	2



(a) Average precision (b) Average relative recall

Figure 7: Average precision and relative recall when $\alpha = 0.5$

4.3.1 Comparison to Similarity-based Ranking

Although semantic information such as task relevance is used in our ranking model, the traditional measures such as precision and recall can also apply when evaluating ranking quality. We therefore compute average *precision* and *relative recall* of all queries.

In general, precision measures the ability of a system to return only relevant results. It is defined as:

$$Precision = \frac{\#\text{ of relevant returned results}}{\#\text{ of all returned results}}.$$

On the other hand, recall measures the ability of a system to return *all* relevant results, and is defined as:

$$RelativeRecall = \frac{\#\text{ of relevant results}}{\#\text{ of relevant results returned by all methods}}.$$

It is a common practice to focus only on the top-k results. In the experiment we follow the practice to use the top-10 results only. Another important aspect is when calculating the recall measure, we need to know the total number of relevant results, which is extremely difficult to know. We use the total number of unique relevant results returned by the two ranking schemes. For every query, each returns 10 results. so the overall relevant number is 20 at most. That is the base of *relative recall*.

Figure 7 shows the results about the average top-k precision and relative recall for all user queries when $\alpha = 0.5$ (the default setting). Note that the results returned by the personalized ranking method will be affected by the current state (i.e., task) of the user. When computing the results for the personalized ranking method, we get the average result for all the tasks. From the results, we can see that the personalized ranking method outperformed the traditional method. That is because our ranking results take into consideration not only the query terms but also the context of that user. The number of relevant results returned by our method is larger than that of the similarity-based ranking method in Lucene. So the ranking quality is improved.

4.3.2 Effect of User Model

The major advantage of our ranking scheme is that it could utilize the desktop user model to track the user's current desktop activity and then use this information to predict the desktop resources that the user is most likely to access. These resources will be moved to a higher rank. Due to the lack of standard desktop benchmark datasets, it is very difficult to give theoretical or experimental analysis to the quality of the ranking result in a quantized way. Here

Table 1: Tasks discovered after refinement

Task	Resources involved	Brief description
T1	Documents/Beagle/RDFRepository/RDFRepository_JavaDoc.html Tools/openrdf-sesame-2.1.3/docs/users/ch07.html Tools/openrdf-sesame-2.1.3/docs/users/ch08.html Materials/Lucene_in_ Action.pdf Documents/Beagle/Minack 2008_LuceneSail.pdf Materials/RDF/RDF_Query_Language_Compare.htm Tools/openrdf-sesame-2.1.3/docs/users/index.html Tools/openrdf-sesame-2.1.3/docs/users/ch09.html Tools/openrdf-sesame-2.1.3/docs/users/ch06.html	*Lucene* and *Sesame* studying
T2	Papers/PageRank/RandomWalks.ppt Papers/PageRank/The PageRank Citation Ranking- Bringing Order to the Web (1998).pdf Downloaded/auth.pdf Papers/PageRank/Inside PageRank.pdf Papers/PageRank/ObjectRank.pdf	Survey of popular ranking schemes
T3	Documents/Beagle/Beagle_Doc/Backend_Tutorial.htm Materials/BeagleInvestigation/BeagleInv.pdf Documents/Beagle/Beagle_Doc/Indexing_Data.htm Documents/Beagle/Beagle_Doc/Architecture_Overview.htm Documents/Beagle/WhitePapers/Beagle++ - Indexing and Querying your Desktop.pdf Documents/Beagle/Beagle_Doc/Filter_Tutorial.htm	Investigation of *Beagle*'s indexing module
T4	Papers/SemanticDesktop/chi08-feldspar.pdf Slides/Comparions of Desktop Search Tools.ppt Papers/Beagle++/ESWC06- Beagle++.pdf Slides/semantic search.ppt Slides/Beagle++Toolbox.ppt Papers/Beagle++/ESWC05-Activity-Based Metadata for Semantic Desktop Search.pdf	Investigation of several desktop search tools
T5	Documents/Beagle/WhitePapers/Beagle++Ranking.pdf Documents/Beagle/Beagle_Doc/Searching_Data.htm Documents/Beagle/Beagle_Doc/Backend_Tutorial.htm Materials/BeagleInvestigation/BeagleInv.pdf Documents/Beagle/Beagle_Doc/Architecture_Overview.htm Papers/PageRank/ObjectRank.pdf	Investigation of *Beagle*'s ranking module

we first demonstrate the effect of our desktop activity predication model by showing a real case found when analyzing user $A's$ survey form. Then, we investigate the impact of the tuning factor α to the ranking results.

As shown in Table 3, we found that user A issued the query "*PageRank model*" two times in the second session of the experiment. His descriptions about his needs at the time of querying were: first, get some ideas about the technology in Beagle++. Second, compare the popular ranking algorithms including PageRank and HITS.

The first task is about the technology used in *Beagle++*. User A submits the query to find documents about the implementation of the PageRank in Beagle++. The second one is to investigate popular ranking algorithms. It is about the model, features of different ranking methods. Obviously user A wants different results but he uses the same query.

Our personalized ranking method clearly distinguishes the two queries. The ranking results ($\alpha = 0.8$) are then shown in Table 3. To save space, only the top 5 results are listed in each case. Both the ranking results from Lucene and the results from the personalized ranking are listed. As for our method, both the final scores and the relevance scores are listed.

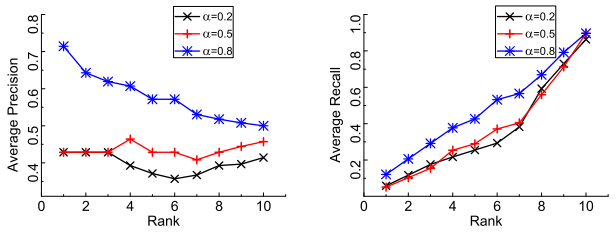
The first results from the personalized ranking (when the user was in state T1) is exactly the same as the ranking results from Lucene. This is due to the fact that no resources containing the query keywords have a high probability score when the current user state is T1. But when the user is in state T2, our results are much more reasonable than Lucene's results. Resources about different ranking algorithms are moved far ahead in the ranking list. Table 3 shows that the probability scores of these resources are quite high while their similarity scores (Lucene's score) are relatively low. Our method produces a balanced list which are more favorable to the user.

This example shows that our model understands the user's real needs better by knowing the context of the user at the querying time. Traditional desktop search applications only rely on relevance to keywords to rank resources. This simple ranking schema does not work well sometimes. In the above example, Lucene performs bad because the keyword "*model*" is a common word. It appears in many documents. Lucene can not tell the relevant documents from the large number of candidates only by similarity scores.

We further investigate the impact of the tuning factor α to the ranking results. As shown in Figure 8, when α increases, i.e., the weight of the relevance score is increased, the precision is significantly improved and the recall is also slightly improved as well. This means our ranking scheme is more effective in "guessing" the real needs of the user when he issues a query.

Table 3: The case of user A

Ranking Scheme	Ranking Result List	Lucene score	Our score (relevance score)
Lucene	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	
	openrdf-sesame-2.1.3/docs/system/ch06.html	0.709	N/A
	Jena-2.5.5/doc/images/Ont-model-layers-import.png	0.616	
	Jena-2.5.5/doc/images/Ont-model-layers.png	0.616	
HMM (T1)	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	0.187(0.010)
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	0.165(0.010)
	openrdf-sesame-2.1.3/docs/system/ch06.html	0.709	0.150(0.010)
	Jena-2.5.5/doc/images/Ont-model-layers-import.png	0.616	0.131(0.010)
	Jena-2.5.5/doc/images/Ont-model-layers.png	0.616	0.131(0.010)
HMM (T2)	The PageRank Citation Ranking- Bringing Order to the Web (1998).pdf	0.142	0.305(0.345)
	Inside PageRank.pdf	0.427	0.272(0.233)
	RandomWalks.ppt	0.267	0.212(0.199)
	Jena-2.5.5/doc/javadoc/com/hp/hpl/jena/rdf/model/Model.html	0.896	0.179(0.0004)
	openrdf-sesame-2.1.3/docs/system/ch03.html	0.786	0.158(0.0004)



(a) Average precision (b) Average relative recall

Figure 8: Average precision and relative recall with different α

5. RELATED WORK

Some previous researches try to build client side information collector for user modeling. Mitchell et al. [6] try to extract user knowledge from raw workstation contents such as emails and local files. Shen et al. [8] focus on implicit user feedbacks and email messages. IRIS [3] models the user desktop activities as RDF tuples. However, it requires specially instrumented desktop applications. PLUM [5] is another ubiquitous user information collector for Mac OS and Windows. As IRIS, the user activities are also modeled into RDF tuples. Different from other work, PLUM observes a large range of user information, including running processes, nearby WIFI access point, application-specific activities, etc. These user information collector can be used directly in our PaaS architecture.

6. CONCLUSION

This paper introduces PaaS, an architecture of personalized services for the cloud age. In PaaS, there are four important modules in the client side. The user information collector gathers the user's personal information. The user model builder constructs the user model according to the information and keep it in the user model store. The model synchronizer serves cloud side services by executing their queries for user models. We implemented a cloud data search prototype as our case study. The user is able to enjoy much better personalized experience with our prototype.

7. REFERENCES

[1] M. Armbrust, A. Fox, and R. G. et al. Above the clouds: A berkeley view of cloud computing. Technical report, Electrical Engineering and Computer Sciences Univeristy of California at Berkeley, 2009.

[2] J. Chen, H. Guo, W. Wu, and C. Xie. Search your memory ! - an associative memory based desktop search system. In *Proceedings of the ACM SIG conference on Mangement of Data*, 2009.

[3] A. Cheyer, J. Park, and R. Giuli. Iris: Integrate. relate. infer. share. In *1st Workshop on The Semantic Desktop. 4th International Semantic Web Conference*, 2005.

[4] D. Kelly and J. Teevan. Implicit feedback for interring user preference: a bibliography. In *ACM SIGIR Forum*, volume 37, pages 18–28, 2003.

[5] M. V. Kleek and H. E. Shrobe. A practical activity capture framework for personal, lifetime user modeling. *Lecture Notes in Computer Science*, 4511, 2007.

[6] T. Mitchell, S. Wang, Y. Huang, and A. Cheyer. Extracting knowledge about usersąŕ activities from raw workstation contents. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[7] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989 1989.

[8] J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *Proceedings of the 11th international conference on Intelligent user interfaces*, 2006.

[9] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[10] G. Xue, H. Zeng, Z. Chen, and Y. Y. et al. Optimizing web search using web click-through data. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 118–126, 2004.