

# Architectural Design of WebScales - A Large-Scale Metasearch Engine

Weiye Meng  
SUNY at Binghamton  
Department of Computer Science  
Binghamton, NY 13902  
1-607-777-4311  
meng@cs.binghamton.edu

Clement Yu  
University of Illinois at Chicago  
Department of Computer Science  
Chicago, IL 60607  
1-312-996-2318  
yu@cs.uic.edu

Zonghuan Wu, Vijay Raghavan  
University of Louisiana at Lafayette  
Center for Advan. Computer Studies  
Lafayette, LA 70504  
1-337-482-5243  
{zwu,raghavan}@cacs.louisiana.edu

## ABSTRACT

It is estimated that there are hundreds of thousands of information sources on the Web, including both the Surface Web and the Deep Web. Most of these sources have their own search capabilities. In order to alleviate ordinary users from the formidable task of identifying useful sources and search them individually, it is important to provide a unified access to these sources. Metasearch engine is such a system that can provide unified access to multiple existing search systems. In this paper, we provide an architectural design of a large-scale metasearch system, WebScales. We also discuss what we have already done in developing an operational system based on the proposed architecture.

## Categories and Subject Descriptors

C.2.4 [Computer-Communication Networks]: Distributed Systems - distributed databases.  
H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval - search process; selection process.  
H.3.4 [Information Storage and Retrieval]: Systems and Software - information networks.

## General Terms

Algorithms, Design

## Keywords

Metasearch, search engine, distributed digital library, WebScales

## 1. INTRODUCTION

The World Wide Web has been considered as the largest digital library in recent years. People all over the world use the Web to find all sorts of information, from medical information to food recipes. The Web is inherently distributed and the data on the

Web is stored in numerous sources. Often, these sources have their search capabilities. For example, many organizations, companies and universities have their own Web sites and most of them have their own search engines.

The information a particular user needs is frequently stored in multiple sources. For example, publications on a subject may be available in the sources of many publishers, universities and research labs. It is difficult for an ordinary user to identify all the sources him/herself and search each source separately to find all information he/she needs. One way to address this problem is to build a metasearch engine on top of multiple sources in advance.

A metasearch engine is a system that supports unified access to multiple existing search engines. Upon receiving a query, it invokes some of the underlying search engines to retrieve suitable documents. It then re-ranks the documents from the different sources and presents the retrieved documents to the user. In comparison to an underlying search engine, the metasearch engine has higher coverage, since its document **database** (i.e., the set of documents searchable by the search engine) is the union of the document databases of all the individual search engines.

The advantages of metasearch engines have been realized by many researchers and practitioners. As a result, many metasearch engines have been created (see, e.g., SavvySearch ([www.search.com](http://www.search.com)), ProFusion ([www.profusion.com](http://www.profusion.com)), Dogpile ([www.dogpile.com](http://www.dogpile.com))) and a large amount of research work has been published in the literature (see Section 3). In this paper, we describe the architectural design of WebScales, a large-scale metasearch engine under development. The main difference between WebScales and other metasearch engines is that the former aims to connect to **all** useful search engines. It is estimated the number of search engines on the Web is in the hundreds of thousands [1]. Therefore, WebScales must emphasize scalability and automation. Existing metasearch engines, in contrast, are small in scale. The current largest metasearch engine, for example, ProFusion, connects to about 1000 search engines.

The rest of the paper is organized as follows. In Section 2, we present the architectural design of WebScales. Different components and their relationships will be discussed. In Section 3, related work with respect to different components will be compared. In Section 4, we present our design and implementation for several components of WebScales. Section 5 concludes the paper.

## 2. WEBSCALES ARCHITECTURE

The architectural design of WebScales is shown in Figure 1. This architecture covers two aspects of the system. The left portion of the figure describes how search engines are identified and incorporated into WebScales and the right portion of the figure describes how a query submitted to WebScales is processed. The components of WebScales are described in more detail below.

1. The **search engine crawler** discovers the search engines on the Web. In order to utilize as many search engines on the Web as possible, it is imperative to discover search engines automatically.

2. The **search engine filter** identifies useful search engines among discovered search engines for inclusion in WebScales. Search engines that should not be used include redundant search engines (whose contents are covered by other selected search engines) and obsolete search engines (i.e., those that are no longer functional). These useless search engines should be identified and filtered out automatically. The search engine filter has a subcomponent called **search engine categorizer** that categorizes search engines into appropriate topics according to their contents. When search engines are categorized, it is easier to identify redundant search engines and it also helps with identifying useful search engines to invoke for a particular user query (see Section 4.6).

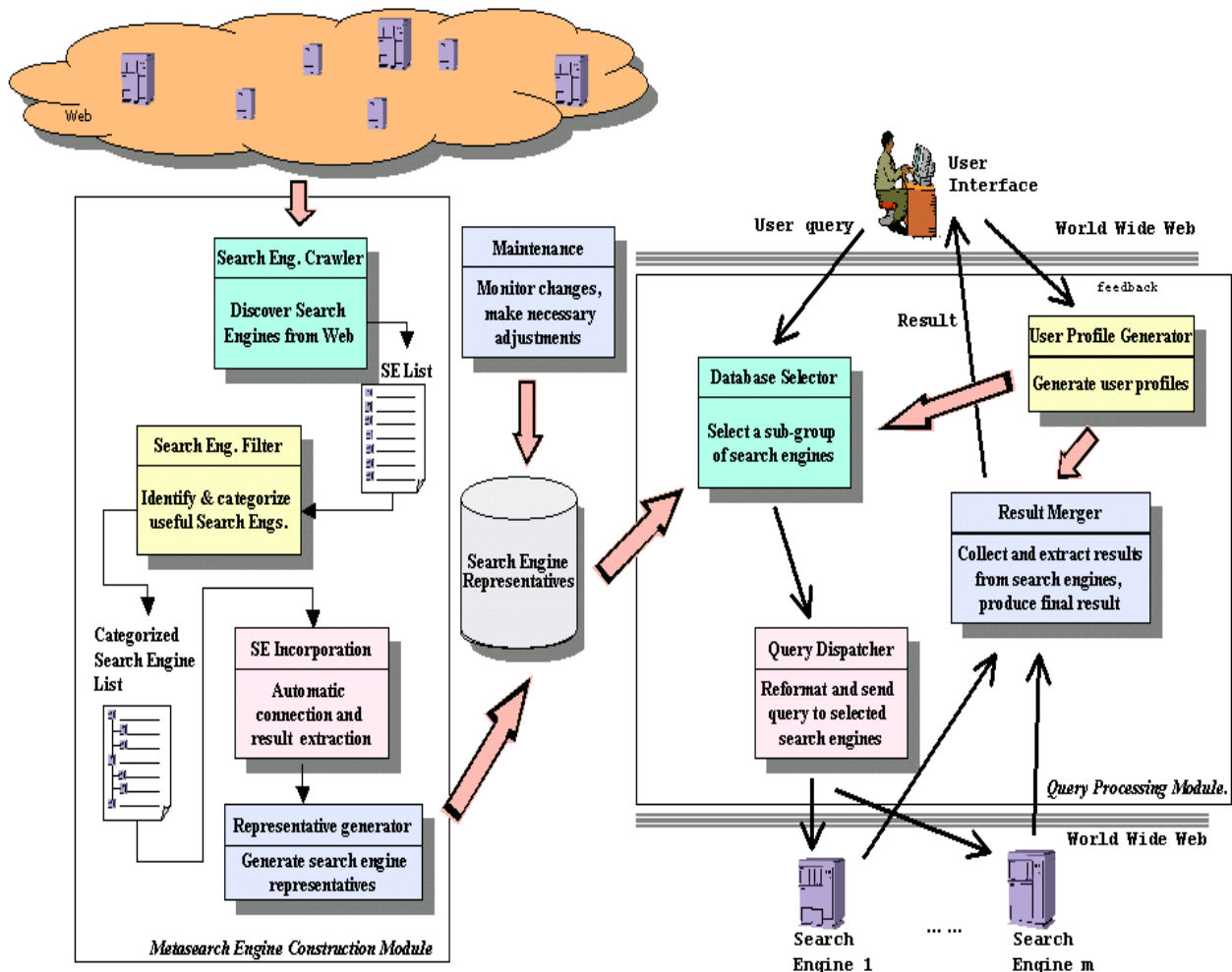


Figure 1: The Architecture of WebScales

3. The **search engine incorporator** incorporates each selected search engine into the metasearch engine. In order to utilize a search engine, a program is needed to automatically connect to it (i.e., pass queries to and receive search results from the search engine). When results are returned from a search engine, another program is needed to extract the useful information (e.g., URLs of retrieved documents and snippets). Due to the number of search engines involved, the connector program and the extractor program must be automatically generated.
4. When a user query is received by the metasearch engine, it is passed to appropriate local search engines. If the number of local search engines is small, it may be reasonable to send the query to all of them. But, if the number is large, say more than one thousand, then sending each query to all local search engines is no longer a reasonable strategy. Because in this case, a large percentage of the document databases will be useless with respect to the query. Suppose a user is interested in only the 10 best-matched documents for a query. Clearly, the 10 desired documents are contained in at most 10 databases. Consequently, if the number of databases is much larger than 10, then a large number of databases will be useless with respect to this query. Sending a query to the search engines of useless databases has several problems. First, dispatching the query to useless databases wastes the resources at the metasearch engine site. Second, transmitting the query to useless search engines from the metasearch engine and transmitting useless documents from these search engines to the metasearch engine would incur unnecessary network traffic. Third, when a query is evaluated against useless local databases, resources at these local systems would be wasted. Fourth, when a large number of documents are returned from useless databases, more effort would be needed by the metasearch engine to identify useful documents. Therefore, it is of great importance to send each user query to only potentially useful search engines. The problem of identifying potentially useful search engines to invoke for a given query is known as the **database selection problem**. The **database selector** component is responsible for performing database selection.
5. In order to perform database selection, characteristic information that reflect approximately the contents of the database of each search engine needs to be collected. This information is called the **representative** of the database or search engine. Database selection is performed by comparing a user query with the database representatives. The representative generator is responsible for generating the database representatives.
6. To improve the retrieval of useful information for users, the **user profile generator** keeps track of the retrieval activities of users and uses the information to create user profiles. User profiles can be used to support personalized retrieval. In Section 4.5, we will provide a method to perform database selection using user profiles.
7. The **query dispatcher** sends the user query to selected search engines. The query needs to be formatted properly for each search engine based on the required query format of the search engine. The function of the query dispatcher is provided by the search engine incorporator. Therefore, this component will not be further discussed in this paper.
8. After the results from selected search engines are returned to the metasearch engine, the **result merger** combines the results into a single ranked list. Only a desired number of the top-ranked documents in the list are forwarded to the user interface to be displayed.
9. Search engines may change over time and these changes may affect its use in a metasearch engine. For example, a search engine may change its format for accepting user queries and such a change may cause the search engine unusable in the metasearch engine unless the metasearch engine's connection program for the search engine is updated accordingly. As another example, a search engine may update its database (e.g., by indexing new documents) and the change needs to be reflected in the representative information maintained by the metasearch engine about this search engine. Therefore, it is necessary to maintain the metasearch engine to reflect the changes made by local search engines. Maintenance is made easier in WebScales due to the employment of automated techniques. For example, when our metasearch engine fails to connect to a previously connectable search engine due to the changes in its query method, our automatic search engine incorporation program can be triggered to refetch the interface page of the search engine and update the corresponding connection program automatically. Metasearch engine maintenance issues will not be further discussed in this paper.
10. It is possible to include "value-added" components to this architecture for increased retrieval effectiveness. For example, a "training collection" which has similar coverage of subject matters and terms as the collection of databases to be searched may be utilized [40]. Upon receiving a query, the training collection is searched, terms are extracted and then added to the query before retrieval of documents from the actual collection takes place. As another example, if linkage information among Web pages is available, then PageRank that measures the global importance of pages can be computed [2] and this information can be utilized to select databases and retrieve documents for better retrieval [45]. "Value-added" components will not be discussed further in this paper.

In Section 4, we describe in more detail our design for all of the above components except components 7, 9 and 10 (our research on components 9 and 10 is on-going and details are not finalized).

### 3. RELATED WORK

Metasearch engine and distributed information retrieval have been active research areas in recent years (please see [28] for a recent survey of the research activities). While the query processing modules (i.e., the right portion of Figure 1), especially the database selection and result merging components, have been investigated extensively, little attention has been paid to the automatic construction of large-scale metasearch engines. Current metasearch engines are small in scale (most connect to less than 20 search engines). As a result, issues such as automatic discovery of search engines, automatic identification of useful search engines and automatic incorporation of search engines into a metasearch engine were not critical to these small-scale

metasearch engine. However, as the goal of WebScales is to connect to as many useful search engines as possible, it is necessary to address these issues.

We now compare our approach with related work for each component.

1. We are not aware of any published work on automatically discovering search engines. CompletePlanet ([www.completeplanet.com](http://www.completeplanet.com)) claims to have developed techniques to automatically identify search engines. Similarly, InvisibleWeb ([www.invisibleweb.com](http://www.invisibleweb.com)) has designed crawling based semi-automatic method to discover Web interfaces of searchable databases. However, details of both techniques are proprietary and are not publicly available.
2. To the best of our knowledge, there is no published work specifically addressing the issue of automatically identifying all useful search engines. However, research on automatic categorization of search engines has been reported [18][26]. The technique reported in [18] is very different from ours [26]. The former uses the number of hits information while we use the similarity of retrieved documents (see Section 4.2.1 for a more detailed comparison). CompletePlanet also employs an automatic search engine categorization technique to organize search engines but the technique is not published.
3. ProFusion ([www.profusion.com](http://www.profusion.com)), CompletePlanet and InvisibleWeb all claim that they have technologies to connect to search engines automatically or semi-automatically, but not much detailed information is available to the public. Search engine result extraction is a special case of the more general Web data extraction problem. Again, details of the data extraction approaches applied to metasearch engines such as ProFusion, SavvySearch ([www.search.com](http://www.search.com)) and search engine directory providers such as CompletePlanet and InvisibleWeb are lacking.
4. Database selection is one of the most researched areas in metasearch engine and distributed information retrieval research (see for example [5], [10], [13], [14], [16], [17], [25], [27], [37], [44]). Our approach (see Section 4.4) differs from other approaches in the following aspects. First, a special statistic called maximum normalized weight is used in our approach but not in other approaches. Our investigation indicates that this statistic is critical in accurate database selection [25]. Second, our approach is based on a proved theory of ranking databases optimally [44]. Third, our approach of combining all database representatives into a single database representative while discarding non-essential information is particularly suited to deal with a huge number of search engines [27].
5. All database selection techniques utilize some sort of database representatives and face with the same problem of how to generate their needed representatives. Almost all existing techniques assume that the entire document collection of a database can be accessed for generating representatives. This is not realistic in the Internet environment. Recently, sampling-based techniques for constructing approximate representatives for uncooperative search engines have been reported [4][19]. Our strategy is

similar to [19], but we estimate different statistics (i.e., maximum normalized weights) of terms that are needed by our database selection technique.

6. In the area of personalized Web search, *WebMate* uses user profiles to refine user queries [6], but no experimental results are given. *Watson* refines queries using a local context but does not learn the user profile [3]. *Inquirus 2* uses users' preferences to select data sources and refine queries but it does not have user profiles, and requires the user to provide his/her preferences of categories [15]. In addition, only four non-topical categories are used in *Inquirus 2*. The method in [32] learns a user's profile from his/her surfing histories, and re-ranks/filters retrieved documents based on the profiles. Our approach differs from all of the above in that we try to map user queries to a small set of topics based on both a user's profile and some general knowledge.
7. Result merging is another extensively studied area in metasearch engine. Most proposed approaches use the local scores/ranks of the retrieved documents to compute the overall scores/ranks of these documents [5], [10], [14], [35], [38], [46]. Similar to the *Inquirus* system [21], our approach ranks retrieved documents using global similarities. Please see Section 4.7 for the advantages of using global similarities for ranking.

## 4. DESIGN OF COMPONENTS

In this section, we provide a description for each component (except components 7, 9 and 10) introduced in Section 2.

### 4.1 Search Engine Crawler for Search Engine Discovery

It is reasonable to assume that the interfaces of all search engines are crawlable on the Web even though pages searchable from some of these search engines (in particular those in the Hidden Web) are not crawlable. We designed a special Web robot to discover search engines. It is similar to regular Web robots used to collect Web pages but with the capability to determine if a crawled page is the interface page of a search engine. When a page is returned by the special robot, a set of rules is applied to determine if the page has a search engine interface on it. We have implemented an operational Web robot for discovering search engine pages. Among the rules used, two main rules are: (1) The HTML source file of the page contains a **form** tag with certain attributes such as "method" and "action". For example, the form tag of the ACM Digital Library search engine (<http://www.acm.org/dl/Search.html>) is `<FORM METHOD="POST" ACTION="/ows-bin/dl/owa/dl_srch.new"> ... </FORM>`. (2) At least one of the three key words "search", "query" and "find" appears in the form tag or at a location nearby. Preliminary experimental results from crawling the Web pages of 8 universities indicate these rules are very effective (218 search engine pages were identified with a correctness rate of about 94%).

Note that the special robot should be run periodically to discover search engines that are newly available since the previous run.

## 4.2 Search Engine Filter

This component is to identify redundant and obsolete search engines so as to exclude them from the metasearch engine. A search engine is redundant if its document set is covered by another search engine that is used by the metasearch engine. For example, an organization may have an organization-wide search engine and one division of the organization may have a division-wide search engine. In this case, the database of the latter can be contained in the database of the former. Mirror sites are a special case of redundant search engines. Our methodology for identifying redundant search engines is as follows. First, a topic hierarchy is constructed. This can be obtained from an existing hierarchy (such as the Yahoo category hierarchy or the Open Directory hierarchy) or by modifying an existing one. Next, the search engines are assigned to appropriate topics in the hierarchy based on their contents (see Section 4.2.1 for a description of our database categorization algorithm). Third, the search engines assigned to the same topic are grouped into a subset. This reduces the number of search engine pairs to be examined. Finally, the documents found by the search engines in each subset are examined to identify redundant search engines. The detection of whether a search engine A is covered by another search engine B can be carried out as follows. First, a set of documents from A is randomly retrieved from A. Next, for each such document, say  $d$ , we form a special query consisting of many terms that appear frequently in  $d$ . Because of how the query is formed, any search engine containing  $d$  should return  $d$  as one of the top-ranked documents. Now, submit the query to B to see if the URL of the document is returned. If each document in the random set returned by A is also retrieved by B, then we declare that A is covered by B. Otherwise, A is not covered by B.

Obsolete search engines are easy to detect, because when a query is submitted to such a search engine, an error message indicating that the server is no longer accessible will be displayed.

### 4.2.1 Search engine categorizer

The algorithm used by this component was presented in [26], [39] and was called HASRD (High Average Similarity of Retrieved Documents). In this section, we provide a brief summary of the algorithm.

First, a **description** of each topic is generated. The description contains the terms that appear in the topic as well as in the child topics of the topic. Such a description is represented as a vector of terms with weights. The weight of each term is computed using the widely used  $tf*idf$  formula [34], where  $tf$  is computed based on the number of times a term appears in the description and  $idf$  is computed based on the number of descriptions in the same level of the topic hierarchy that contain the term.

Next, each description vector is used as a query and is submitted to each search engine. The weights of the terms in a description vector are reflected in the query by repeating them appropriate number of times proportional to the relative weights, i.e., terms with higher weights are repeated more times. When a query is submitted to a search engine, the top  $N$  returned documents are fetched, for some  $N$ , and their similarities with the query are computed based on a global similarity function. Finally, the average of these similarities is taken and is treated as the similarity between the query and the search engine.

Finally, for each search engine, sort the topics in non-ascending similarities between all topics (their description queries) and the search engine, and assign the search engine to the top  $k$  concepts for some integer  $k$ . If a search engine is assigned to a high percentage of the child topics of a topic, then it is assigned to the parent topic directly (no longer assigned to the child topics).

It is clear that this algorithm assigns highly specialized search engines to the lower-level (i.e., more specific) topics and more general search engines to higher-level topics. In particular, general search engines such as Google and AltaVista will be assigned to the root topic of the hierarchy, indicating they return high quality documents for almost all topics.

A different approach, known as PnC (Probe-and-Count) to categorize search engines is reported in [18]. There are mainly two differences between PnC and HASRD. The first is that PnC uses trained topic queries for search and the second is that it uses the number of hits of each search engine with respect to each query to perform categorization. Intuitively, if a large number of documents can be retrieved from a search engine for a given topic query, then the search engine has a good coverage of the topic. PnC is more efficient than HASRD as, unlike HASRD, it does not require downloading of retrieved documents. However, PnC may have a limitation. Many Web search engines do not return the number of hits for user queries and PnC cannot be used for these search engines.

Intuitively, both the average similarity and the number of hits are useful for search engine assignment. With respect to a given topic, the former measures the **quality** of the (top) documents in a search engine while the latter reflects the **coverage** of the documents in a search engine. Therefore, when both the average similarity and the number of hits are used, more accurate assignment is expected. We are investigating this combination.

## 4.3 Search Engine Incorporator

This component aims to incorporate a search engine into a metasearch engine automatically when the URL of the search engine is provided. As mentioned earlier, for each selected search engine, this component needs to automatically generate two programs, one for connecting to the search engine and the other for extracting the result information from the returned result page by the search engine for each query.

### 4.3.1 Automatic search engine connection

This includes correctly passing queries to and getting result pages back from each search engine. This can be achieved by analyzing the source HTML file of a search engine page. **Form** tags with the attributes **method** and **action** are of special interest. Specifically, from the method attribute, the HTTP request method (mostly GET and POST) supported by the search engine can be identified and from the action attribute, the search engine server can be located. For example, for the ACM Digital Library search engine, the POST method is supported and the search engine server is at “/ows-bin/dl/owa/dl\_srch.new” which is a relative path to the ACM homepage, i.e., the full path of the server is at [http://www.acm.org/ows-bin/dl/owa/dl\\_srch.new](http://www.acm.org/ows-bin/dl/owa/dl_srch.new) and the CGI program is dl\_srch.new. The CGI program is used to pass a query to the search engine and activate it and send back results returned by the search engine. The form tag usually has a nested input tag

with a name attribute which identifies the query name for passing queries to the search engine server. Again, for the ACM Digital Library search engine, we have <FORM ... <INPUT TYPE="text" NAME="fields\_expr" size="32" class="textfield" ... </FORM> and the query name is "field\_expr". With the information regarding HTTP request method, the full path to the CGI program and the query name available for a search engine, a connection program for the search engine can be automatically generated. Our preliminary implementation and evaluation of the program has been carried out. Among the 190 commercial search engines listed at site www.searchenginewatch.com that we used to test our program, we achieved a success connection rate of 93%. We are working on to further improve the accuracy.

### 4.3.2 Automatic result extraction

When a query is submitted to a search engine, result pages that contain the URLs of retrieved documents (and possibly snippets) are generated. Usually, a result page only contains a small number of (say 10) result URLs and more result URLs can be obtained by clicking additional links in the result page. Also, a result page usually contains un-related URLs (of advertisement pages and/or maintenance/service pages of the search engine). The problem is how to obtain a given number of URLs of retrieved documents starting from the first result page. Since we are dealing with a huge number of search engines, we need to develop a general result extractor that works for any search engine.

In this paper, we focus on how to obtain the URLs of retrieved documents only. Our approach consists of the following two steps.

1. An "impossible query" (a query consisting of a single non-existent term) is sent to the search engine under consideration. This will result in a "Result Not Found" page to be sent back. For example, we can send "SendImpossibleQueryResultCanNotBeFound" as a query to a search engine and it is very unlikely that a search engine has any document containing this term. From the search engine returned page, we know that there are no result URLs, only useless (useless to the metasearch engine) URLs such as advertisement URLs or service URLs. We record these URLs, so, when these URLs appear in returned result pages for other queries, we can easily exclude them. The Layout pattern of the "Result Not Found" page is also recorded for future reference. The same idea is used in [9] (they called it sending "dummy query" to search engines) to build the ShopBot, a comparison-shopping agent on the Web.
2. Several sample queries are sent to the search engine to obtain sample returned result pages. From these HTML pages, after excluding the URLs appearing in the result page of the "impossible query", tag patterns associated with the remaining URLs are generated. Since each sample query is likely to yield a number of result URLs and the same search engine usually presents its results in the same style, the tag pattern with the highest frequency is usually the pattern for the result URLs for the search engine. After this pattern is identified, it is used to extract the result URLs for future queries submitted to this search engine.

Our evaluation indicates that if good sample queries (i.e., they can yield significant results) are used, for over 95% of the 78 search

engines we tested, the above method can extract correct result URLs.

## 4.4 Database Selector

In this section, we report our database selection techniques [27], [44] designed for the WebScales system.

Suppose the user is interested in retrieving the  $m$  most similar documents (documents with the highest degrees of relevance), where similarities are computed by a global function. The global similarity function can be a standard function such as Cosine [34] or Okapi [29]. It can also incorporate linkage information among Web pages [2], date information as well as other techniques such as utilizing the proximity between query terms in documents to enhance retrieval effectiveness. Databases [D1, ..., Dn] are ranked optimally with respect to a query Q, if for every  $m$ , there exists a  $k$  such that the first  $k$  databases D1, D2, ..., Dk contain the  $m$  most similar documents and each of these databases contains at least one such document. In other words, it is sufficient to retrieve from the first  $k$  databases so as to retrieve all the  $m$  desired documents. A necessary and sufficient condition to rank databases optimally is to rank them in descending order of the similarity of the most similar document in each database [44]. Thus, optimal ranking of databases can be performed. The optimal ranking condition cannot be used as is because we cannot afford to actually search each database to obtain the similarity of its most similar document. Thus, we estimate the similarity of the most similar document in each database based on the representatives of search engines and then rank databases (search engines) in descending order of these similarities.

We now describe how to estimate the similarity of the most similar document in a database D using its database representative and the query. The representative of a local database consists of all terms in the database. For each term, say term  $t_i$ , two quantities are kept. They are the maximum normalized weight,  $mnw_i$ , and the average normalized weight,  $aw_i$ . Conceptually, the documents in all local databases can be considered as belonging to a global database. A representative for this global database is also maintained. The representative of the global database consists of all terms appearing in any of the local databases and for each term, only one quantity, namely the (global) document frequency of the term (the number of documents containing the term), is kept. This generates a weight known as the global inverse document frequency weight of the term and is denoted by  $gidf_i$  for term  $t_i$ . The  $gidf_i$  weight of a term is a decreasing function of the document frequency of the term [34].

Let  $(q_1, \dots, q_t)$  be the vector of a query Q with  $t$  terms. The similarity of the most similar document of a database D with respect to Q is estimated by:

$$msim(Q, D) = \max_{1 \leq i \leq t} \left\{ \sum_{j=1, j \neq i}^t q_j * gidf_j * aw_j + q_i * gidf_i * mnw_i \right\} / |Q| \quad (1)$$

where  $|Q|$  is the norm of the query vector. Since the query norm is common to all databases, it can be omitted. The intuition for the estimate is as follows. The most similar document in a database can be a document having the maximum normalized term weight of the  $i$ -th query term. If that is the case, the document is likely to

have the average term weight for each of the other query terms. Since the  $i$ -th term can be any one of the query terms, we take the maximum over all query terms. It is easy to see that the estimation can be carried out in time  $O(t)$ , where  $t$  is the number of query terms and therefore the process is very efficient. It can also be shown that the estimation is 100% accurate for single-term queries and therefore optimal results are guaranteed for those queries. This is significant as about 30% of all Internet queries are single-term queries [20]. More details about this estimation method are provided in [44].

As WebScales is designed to connect to hundreds of thousands of search engines, the above database selection method may take too long as each database needs to be scored for a given user query. In other words, its computation efficiency may not scale well. In addition, the storage requirement for all database representatives may be excessive. Although storage media is becoming increasingly affordable, excessive space requirement may force a large portion of the database representatives to be stored on slower storage devices (e.g., disk), causing the database selection process to be slowed down. Our integrated representative approach [27] tackles both problems.

This approach consists of the following steps.

1. Reduce the size of the database representative of each database. We observed that the maximum normalized weight of a term is typically much larger than the average normalized weight of the term as the latter is computed over all documents including those that do not contain the term. This feature implies that in formula (1), if all query terms have the same term frequency weight (a reasonable assumption as in a typical query, each term appears once),  $gidfi * mnw_i$  is likely to dominate the sum, especially when the number of terms,  $t$ , in a query is small (which is typically true in the Internet environment). In other words, the rank of a database  $D$  with respect to a given query  $Q$  is largely determined by the value of

$$\max_{1 \leq i \leq t} \{q_i * gidf_i * mnw_i\}$$

Based on this observation, we introduce a new database representative that, for each term  $t_i$  in database  $D$ , contains only one quantity, namely the adjusted maximum normalized weight  $ami = gidfi * mnw_i$ . With this representative, the global database representative is also no longer needed.

2. Create an integrated database representative for all databases. For a given positive integer  $r$  and term  $t_i$ , the  $r$  largest adjusted maximum normalized weights and the ids of the sites containing these weights are stored in the integrated representative. The idea is to store only important database information for each potential query term. One way to determine the value  $r$  is as follows. If the metasearch engine is designed to search no more than  $k$  search engines for any given query (a small  $k$ , say 20, will be sufficient for most users if relevant search engines can be selected), then  $r$  can be set to  $k$ . This integrated representative can scale to virtually unlimited number of component databases in terms of storage. In [27], it is estimated that the size of the integrated representative is less than 2 GB, regardless of how many component databases are employed. 2 GB is well within the memory capacity of a well-equipped server.

Intuitively, with the integrated representative, the number of pieces of information stored for each term is a constant factor of  $r$  and is independent of the number of databases. In contrast, in non-integrated representatives, the number of pieces of information stored for each term is a constant factor of the number of databases.

3. Achieve good computation efficiency. When evaluating a user query  $Q$ , we compute the ranking scores for only those databases that are among the databases with the  $r$  largest adjusted maximum normalized weights for the query terms. Thus, for a query  $Q$  having  $t$  terms, at most  $t * r$  ranking scores are computed. This is independent of the number of databases. In the Internet environment,  $t$  is usually very small ( $t = 2.2$  on the average [21]). The value of  $r$  is also a small constant (see the previous item). As a result, our proposed method is also highly scalable in terms of computation.

## 4.5 Representative generator

The problem of generating the database representative for an uncooperative search engine consists of the following two steps.

1. Obtain a vocabulary for the database of the search engine. This is to find the most representative terms of the document collection. Our technique for this step is similar to that proposed in [19]. Suppose search engines have been assigned to topics as discussed in Section 4.2.1. To obtain the approximate vocabulary for a search engine, the terms related the topics of the search engine are used as queries and are submitted to the search engine. Then the vocabulary of the search engine is constructed from the terms in the documents retrieved by these queries.
2. Obtain the needed statistics for each term in the vocabulary. As mentioned previously, our database selection method requires a special statistic called the maximum normalized weight for each term. In [24], we presented a method to obtain the approximate statistics needed by our approach. The basic idea is to submit terms in the vocabulary as queries to the search engine under consideration and to estimate the maximum normalized weights based on the top documents returned by the search engine.

After obtaining the representatives for individual search engines, we integrate them into a single integrated representative for better storage and computation efficiency (see Section 4.4).

## 4.6 User profile generator

Successful information finding on the Web is sensitive to the background interests of individual users. For example, for the query "apple", a person with a history of retrieving documents in the computer area is more likely to be interested in information related to "Apple computer" while a person interested in food probably would like to see pages that consider apple as a fruit. Personalized search is an important method to improve the retrieval of relevant documents. The background information of each user (i.e., user profile) can be collected in a number of ways such as through the use of bookmarks and cookies.

Parts of user profiles can also be generated from implicit user feedback. When a user submits a query to a (meta)search engine, the user is returned a number of documents. Typically, the user

clicks some promising links to take a closer look at the documents. If a document is of interest to the user, the user takes some time to read it. Parts of a user profile can be generated from the significant terms in the documents that the user finds promising.

The database selection method described in Section 4.4 aims to retrieve the most similar documents across multiple search engines. One way to improve the retrieval of useful documents (not just similar documents) is to use topics as context to disambiguate the meanings of terms and by selecting the topics based on the use profile. In Section 4.2.1, we introduced a method to assign search engines to appropriate topics. For a user query, if we can first identify the topics related to the query based on the user's profile and use only the search engines assigned to these topics to evaluate the query, then retrieval effectiveness can be improved. More specifically, we can perform a two-step database selection. In the first step, we identify the search engines assigned to the topics related to the user's query, and in the second, we select the best-matching search engines returned from the first step using the method described in Section 4.4.

In [23], we proposed a technique to automatically identify appropriate topics for a given user based on his/her profile. This method is outlined below. Construct a matrix,  $QT$ , whose rows are the queries previously submitted by the user and some of the relevant documents to these queries. The columns are the terms used in these queries plus significant terms extracted automatically from the relevant documents. The  $(i,j)$ th entry of the matrix denotes the weight of the  $j$ th term in the  $i$ th query/document. Construct another matrix,  $QC$ , whose rows are identical to and in the same order as the queries/documents appearing in the matrix  $QT$ . The columns are the various topics in the topic hierarchy. (It is possible to obtain such information without explicit user involvement, as pointed out in [23]). The  $(i,j)$ th entry of the matrix  $QT$  is 1 if the  $i$ th query/document is related to the  $j$ th topic; 0, otherwise. From these two matrices, we can construct a matrix  $TC$  which measures the relationships between the terms (appearing as rows of  $TC$ ) and the topics (appearing as columns of  $TC$ ) such that  $QT*TC \approx QC$ . The approximation is in the least square sense. The matrix  $TC$  can be constructed using a technique similar to that given in [8], [41], [42], by applying Latent semantic Indexing [7], [11]. The matrix  $TC$  can then be used to map a user query to a set of topics.

A user may from time to time submit a query containing words that have never been used by him/her. In that case, it may be desirable to make use of term-topic relationships that have been established from the user profiles of many other users. This can be considered as one form of **collective filtering** in which information that has been found useful to many users is applied to another user. In [23], this is approximated by utilizing a general user profile derived based on a general topic hierarchy (the Open Directory Hierarchy).

We should point out that in addition to being used to determine the topics relevant to a query, user profiles can also be used to refine user queries [3], [6], say by adding important terms in the user's profile to a query, and filter search results [32], say by re-ranking retrieved documents based on their similarities with the user profile.

## 4.7 Result Merger

We plan to implement two different merge algorithms. The first algorithm involves downloading the retrieved documents so as to compute their global similarities. The second algorithm does not require downloading documents but utilizes the ranks of the documents retrieved from multiple databases to arrive at the final ranks of the documents. Only the first algorithm will be described in this paper.

Our algorithm to merge documents retrieved from the invoked search engine is as follows [44]. Suppose the first  $s$  databases have been selected ( $s$  is 2 initially). Each of these selected search engines returns some top ranked documents whose global similarities are computed by the metasearch engine. The global similarity of the most similar document in each of these  $s$  databases is used to compute min-sim that is the minimum of these  $s$  values. Each of the  $s$  search engines then returns to the metasearch engine those documents whose global similarities are greater than or equal to min-sim. If  $m$  or more documents have been returned, then they are sorted in descending order of similarity and the first  $m$  documents are returned to the user, where  $m$  is the number of desired documents specified by the user. Otherwise, the next database will be selected and the above process is repeated until a total of  $m$  documents are retrieved. In practice, in order to avoid invoking the same search engine multiple times for a given query, a sufficient number of top-ranked documents (say  $m$  documents) should be retrieved to the metasearch engine when the search engine is invoked for the first time.

Some of the properties of this algorithm are:

1. If the  $s$  databases containing the  $m$  desired documents are ranked ahead of other databases and the similarity of the  $m$ -th most similar document is distinct, then the algorithm will retrieve the  $m$  most similar documents by searching at most  $s+1$  databases.
2. For single-term queries, all of the  $m$  most similar documents will be retrieved. As about 30% of all Internet queries are single-term queries [20], this result is significant.

We should point out that while downloading documents and analyzing them on the fly can be an expensive undertaking, it also has some clear advantages [22]. First, when trying to download documents, obsolete URLs can be identified. As a result, documents with dead URLs can be removed from the final result list. Second, by analyzing downloaded documents, documents will be ranked by their current contents. Furthermore, if a complicated global similarity function is used, then downloading documents and then computing the global similarities of retrieved documents may be necessary. For example, the proximity of query terms may be utilized in computing the similarities of documents. Database representatives would not incorporate proximity information of terms as it is very inefficient in storage space to record such information.

## 5. CONCLUSIONS

As the number of information sources on both the Surface Web and the Deep Web has reached hundreds of thousands, there is a strong need to build truly large-scale metasearch engines that are capable of utilizing all of these search engines. In this paper, we

provided an architectural design of such a metasearch engine, WebScales, which is under development. When built, WebScales is expected to be able to search far more documents than any current search engines and metasearch engines. Special challenges that arise due to the scale of the metasearch engine were addressed by different components of the architecture. In addition, our current design and implementation of different components were also provided.

## 6. ACKNOWLEDGMENTS

This work is supported in part by the following grants from National Science Foundation: IIS-9902872, IIS-9902792, EIA-9911099, IIS-0208574, IIS-0208434, and Army Research Office ARO-2-5-30267. This work is also supported in part by the Information Technology Initiative of the State of Louisiana to UL Lafayette for the project titled "Information Technology Initiative – Distributed Heterogeneous Information Sources".

## 7. REFERENCES

- [1] M. Bergman. The Deep Web: Surfacing the Hidden Value. BrightPlanet White Paper, <http://www.completeplanet.com/Tutorials/DeepWeb/index.asp>, 2000.
- [2] S. Brin and L. Page. The anatomy of a large-scale hypertextual search engine. WWW 7 Conference, 1998.
- [3] J. Budzik and J. K. Hammond. Watson: Anticipating and contextualizing information needs. 62<sup>nd</sup> Annual Meeting of the American Society for Information Science, 1999.
- [4] J. Callan, M. Connell and A. Du. Automatic Discovery of Language Models for Text Documents. ACM SIGMOD Conference, 1999.
- [5] J. Callan, Z. Lu and B. Croft. Searching Distributed Collections with Inference Networks. ACM SIGIR Conference, 1995.
- [6] L. Chen and K. Sycara. WebMate: A Personal Agent for Browsing and Searching. Autonomous Agents and Multi Agent Systems, 1998.
- [7] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by Latent Semantic analysis. JASIS, 1990.
- [8] R. Dolin, D. Agrawal and A. El Abbadi. Scalable Collection Summarization and Selection. Fourth International Conference in Digital Library, 1999.
- [9] Robert B. Doorenbos, Oren Etzioni and Daniel S. Weld. A scalable comparison-shopping agent for the World-Wide Web. The first International Conference on Autonomous Agents, 1997
- [10] D. Dreilinger and A. Howe. Experiences with selecting search engines using metasearch. ACM Transactions on Information Systems, 15(3), July 1997.
- [11] S. Dumais. Latent Semantic Indexing (LSI) and TREC-2. The Second Text REtrieval Conference, 1994, pp.105-115.
- [12] Y. Fan, and S. Gauch. Adaptive agents for information gathering from multiple, distributed information sources. Proceedings of the 1999 AAAI yposium on Intelligent Agents in Cyberspace, Stanford University, March 1999.
- [13] J. French, A. Powell, J. Callan, C. Viles, T. Emmitt, K. Prey, and Y. Mou. Comparing the performance of database selection algorithms. ACM SIGIR Conference, 1999.
- [14] S. Gauch, G. Wang, and M. Gomez. Profusion: Intelligent Fusion from multiple, Distributed Search Engines. Journal of Universal Computer Science, 1996.
- [15] E. Glover, G. Flake, S. Lawrence, W. Birmingham, A. Kruger, C. Giles, and D. Pennock. Improving Category Specific Web Search by Learning Query Modifications. SAINT, 2001
- [16] L. Gravano and H. Garcia-Molina. Generalizing GLOSS to vector-space databases and broker hierarchies. VLDB Conference, 1995.
- [17] D. Hawking and P. Thistlewaite. Methods for information server selection. ACM Transactions on Information Systems, 17:1, January 1999, 40-76.
- [18] P. Ipeirotis, L. Gravano, and M. Sahami. Probe, Count, and Classify. ACM SIGMOD Conference, 2001.
- [19] P. Ipeirotis, and L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. VLDB Conference, Hong Kong, 2002.
- [20] B. Jansen, A. Spink, J. Bateman, and T. Saracevic. Real Life Information Retrieval: A Study of User Queries on the Web. ACM SIGIR Forum, 32:1, 1998.
- [21] S. Kirsch. Internet search: Infoseek's experiences searching the Internet. ACM SIGIR Forum, 32, 2, 1998, pp.3-7.
- [22] S. Lawrence and C. Lee-Giles. Inquirus, the neci meta search engine. Seventh International World Wide Web Conference, Brisbane, Australia, April 1998), pp.95--105.
- [23] F. Liu, C. Yu and W. Meng. Personalized Web search by mapping user queries to categories. ACM CIKM Conference, 2002.
- [24] K. Liu, C. Yu and W. Meng. Discovering the representatives of a search engine. ACM CIKM Conference, 2002 (poster session).
- [25] W. Meng, K. Liu, C. Yu, W. Wu and N. Rishe. Estimating the usefulness of search engines. IEEE ICDE Conference, 1999.
- [26] W. Meng, W. Wang, H. Sun, and C. Yu. Concept hierarchy based text database categorization. International Journal on Knowledge and Information Systems, 4(2), March 2002.
- [27] W. Meng, Z. Wu, C. Yu and Z. Li. A highly scalable and effective method for metasearch. ACM Transactions on Information Systems, 19(3), pp.310-335, July 2001.
- [28] W. Meng, K. Liu and C. Yu. Building efficient and effective metasearch engines. ACM Computing Surveys, 34(1), March 2002, pp.48-84.
- [29] S. Robertson et al. Okapi at TREC-7: Automatic as hoc, filtering, and interactive track. TREC-7 Conference, 1998.

- [30] M. Mitra, A. Singhal, and C. Buckley. Improving Automatic Query Expansion. ACM SIGIR Conference, 1998.
- [31] M. Montague and J. Aslam. Condorcet fusion for improved retrieval. ACM CIKM Conference, 2002.
- [32] A. Pretschner and S. Gauch. Ontology based personalized search. IEEE ICTAI Conference, 1999.
- [33] I. Ruthvan, C. van Rijsbergen and M. Lalmas. Empirical investigations on query modifications using abductive explanations. ACM SIGIR Conference, 2001.
- [34] G. Salton and M. McGill. Introduction to modern information retrieval. McGraw-Hill, 1983.
- [35] E. Selberg and O. Etzioni. The MetaCrawler architecture for resource aggregation on the Web. IEEE Expert, 1997.
- [36] A. Singhal, C. Buckley, and M. Mitra. Pivoted Document Length Normalization. ACM SIGIR Conference, Zurich, 1996.
- [37] A. Sugiura and O. Etzioni. Query routing for Web search engines: architecture and experiments. WWW Conference, 2000.
- [38] E. Voorhees, N. Gupta, and B. Johnson-Laird, B. Learning collection fusion strategies. ACM SIGIR Conference, Seattle, 1995, pp.172--179.
- [39] W. Wang, W. Meng and C. Yu. Concept hierarchy based text database categorization in a metasearch engine environment. Web Information System Engineering Conference, 2000.
- [40] J. Xu, and J. Callan. Effective Retrieval with Distributed Collections. ACM SIGIR Conference, 1998.
- [41] Y. Yang, and C. Chute. An Application of Least Squares Fit Mapping to Text Information Retrieval. ACM SIGIR Conference, 1993.
- [42] Y. Yang, and C. Chute. An Example-based Mapping Method for Text Categorization and Retrieval. ACM TOIS, 12(3), 1994.
- [43] C. Yu, K. Liu, W. Meng, W. Wu and N. Rishe. Finding the most similar documents across multiple text databases. IEEE Conference on Advances in Digital Library, 1999.
- [44] C. Yu, K. Liu, W. Meng, and N. Rishe. A methodology to retrieve text documents from multiple databases. IEEE Transactions on Knowledge and Data Engineering, 14(6), November/December 2002, pp.1347-1361.
- [45] C. Yu, W. Meng, W. Wu and K. Liu. Efficient and effective metasearch for text databases incorporating linkages among documents. ACM SIGMOD Conference, 2001.
- [46] B. Yuwono, and D. Lee. Server Ranking for Distributed Text Resource Systems on the Internet. 5th International Conference On Database Systems For Advanced Applications, Melbourne, Australia, April 1997.