Developing Buddy: Towards Greater Dependability and Maintainability in Meta-Search

Naresh Kumar Agarwal, National University of Singapore, 3 Science Drive 2, Singapore 117543; E-mail: naresh@comp.nus.edu.sg
Danny C. C. Poo, National University of Singapore, 3 Science Drive 2, Singapore 117543; E-mail: dpoo@comp.nus.edu.sg
Dominick M. T. Leo, National University of Singapore, 3 Science Drive 2, Singapore 117543; E-mail: dominick_leo@comp.nus.edu.sg

ABSTRACT

Most meta-search engines use web scraping as an ad-hoc method to extract results from the output display of various search engine sources. However, a search engine may cease operation, merge with other engines or its display format may change. A dependable meta-search engine must, thus, adapt to display changes in search engine sources and be maintainable even by people with low programming skills. This paper describes the design and development of Buddy, a meta-search engine that is able to help web users search more effectively into multiple search engine sources. It allows integration of a new search source with minimum complexity and programming knowledge, leading to greater dependability and maintainability. Search results are aggregated from multiple sources to remove duplicate and sponsored links and to give the most relevant results each time. Buddy also allows query refinement and saving of search results locally in user computers or remotely in emails.

Keywords: Meta-search engine, Information Retrieval, Web scraping in Java

1. INTRODUCTION

The World Wide Web may be considered the largest database in the world, with its huge collection of data covering every part of our lives. Each day, each second, a humongous number of people search the Web for information and data of their interest, such as news, word documents, research papers, pictures, music and video. The sole aim of these searchers is to find answers to their queries.

However, they may not be able to find all the best answers in a single search engine. E.g. searching Google (www.google.com) alone is still considered insufficient even though it seems to have the largest repository of web pages [1]. This is because there is very little overlap in the databases of different search engines [2]. Since the top results ranked by different search engines are very different from each other, Web searchers potentially miss relevant results by using only one search engine. Here comes the need and relevance of meta-search engines that have the underlying philosophy that "having many heads is better than one" i.e. instead of searching into only one search engine, it may be worthwhile to get another opinion from other search engines. As searching manually into individual search engines is time-consuming and inefficient, meta-search engines (see [3] for a list of meta-search engines) allow searching into various search engines simultaneously.

1.1 Issues with Meta-Search Engines and Their Development

Unfortunately, meta-search engines today are too ad-laden [4]. They are becoming "meta-yellow pages" where searchers query paid listings and get advertisements in their search results. Searchers are forced to sieve through irrelevant sponsored sites ranked among the search results.

There are two ways in which meta-search engines are able to search into other search engines: 1) Using the APIs provided by search engines e.g. Google's Java-APIs. However, this method is not feasible when searching into many search engines. To connect to ten search engines that use different APIs, such a method will require learning how to apply ten different APIs. This will makes the connection to a search engine's database a tedious task and it would be difficult to maintain the system. Moreover, unlike Google, not many search engines are willing to share their APIs with the public. 2) To overcome this limitation, a web scraping technique [5] can be used to extract the results from the output display of various search engines. This allows connecting and extracting data from many search engine sources without having to learn new APIs.

This leads us to the most important issue, which is the focus of this paper – dependability and maintainability. The output display of search engines may change and cause extraction of results using web scraping method to fail. New search engines can emerge anytime in the World Wide Web. Existing search engines may cease to exist or merge with another engine. A dependable meta-search engine must, thus, adapt to display changes in search engine sources and be maintainable even by people with low programming skills.

In this paper, we describe the design and development of Buddy, a meta-search engine developed at the School of Computing, National University of Singapore and accessible at http://buddy.redirectme.net

The remainder of the paper is organized as follows. In Section 2, we briefly describe the features of Buddy that lead to greater maintainability and dependability. Target users and guiding objectives are also discussed. Section 3 discusses the system design considerations. In Section 4, we see the system architecture of Buddy. Section 5 highlights the experimental results on evaluating the system. Section 6 concludes the paper by sharing the lessons learnt and possible future enhancements.

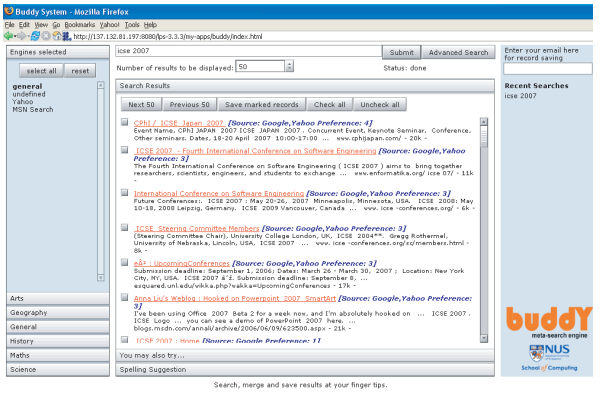
Let us now look at the Buddy Meta-search Engine.

2. THE BUDDY META-SEARCH ENGINE

Figure 1 shows a snapshot of the Buddy Meta-search Engine. Buddy extracts results directly from search engines chosen by the user. No paid links are added into the final merged results. Sponsored links from the source search engines are actually omitted. The system thus minimizes the occurrence of sponsored links in search results, while maximizing relevant links.

Buddy has been designed for dependability and maintainability – it can easily connect or disconnect to/from search engines. At the same time, it can adapt to display output changes in the search engine sources. In Buddy, adding a new source search engine does not require learning its APIs. Any changes to the output display of the source search engines will require minimal modification. The web scraping technique in Buddy makes use of existing Java's Regular Expression and Pattern matching capability [6]. If the output display of source search engines changes, the system administrator just needs to modify the Regular Expression that governs the web scraping structure. This means that there is no need to change the underlying data structures of the system. Maintaining the system will also require little programming knowledge. There is no need to recode the system or web scraping methods if changes occur to the source search engines. Modifications are done in a declarative approach (see Section 3.1).

Figure 1. Snap-shot of Buddy (accessible at http://buddy.redirectme.net)



Understanding searcher needs is important if we are to attract users to use Buddy. Besides being able to extract results from search engines, Buddy can also extract results from sites such as Dictionary.com (www.dictionary.com) to provide spelling suggestions for queries with spelling errors[1].

Buddy can extract query refinement suggestions from sites like Ask.com (www.ask.com) and Yahoo (www.yahoo.com) that provide query refinements together with the searched results. This shows the flexible web scraping method used in our proposed system.

Buddy also enables searchers to save their results locally in their computers or to send their results to their email so that they can access the results in future.

2.1. Target Users

Buddy, though a general-purpose meta-search engine, has been developed keeping in mind, the educational needs of students and teachers in Singapore. The education system in Singapore encourages schools to use materials outside their textbooks, including project-based learning. Earlier, students used encyclopedias to gather this extra information. With the technology available today, students have switched to gathering information from the World Wide Web. Search engines have thus become useful tools for students to do their learning or to gather data for their projects. Teachers can also turn to search engines to gather useful teaching materials. Buddy primarily aims to help such students and teachers in their needs by providing customizable mechanisms to search from specific sources.

Agarwal and Poo [7] discuss classifying a typical Internet searcher into one of 4 searcher modes (or categories) – 1) novice[2], 2) data gatherer, 3) location searcher and 4) focused searcher. In the novice mode, the searcher knows nothing about of the domain under search. As data gatherer, he/she is familiar with the domain or subjects under search. A data gatherer just needs information on the topic he/she is knowledgeable about. A location searcher just needs to locate information previously encountered. The searcher in focused searching mode needs a specific answer to a specific question.

As a novice, it is sometimes hard to decide which results are relevant and which are not. A novice is also unsure of what he/she is searching for. The web-based interface of Buddy has to provide an intuitive way of selecting the search engines. Buddy has separated search engine choices into categories, namely – Science, Math, Geography, History, Arts and General Search. This will helps students to better focus their search into specific domains. Buddy also provides query refinements and spelling suggestions for searchers. This is especially useful when the searcher is in a novice mode. Unlike existing meta-search engines, Buddy does not include sponsored links in its results.

As a data gather, a searcher's aim is to gather information. Searching one search engine is not enough. Searching many search engines manually is inefficient. Buddy is able to search multiple search engines concurrently and return merged results without duplicates or sponsored links. Hence a data gather can select the specific domains he/she wants to search into.

As a location searcher, a searcher wishes to find the results that he/she came across previously. Since Buddy provides utilities to let searchers save or email their results, these can be accessed again locally in user computers or remotely in their email account.

A focused searcher wishes to be able to query about a specific question. Buddy supports Boolean searching. This helps a focused searcher to obtain better results.

As students are still in a stage of learning, we expect more students to fall under the modes of *novice* and *data gatherer*, especially when searching for education-related materials. We would expect teachers to be in the *data gatherer, location searcher*, or *focused searcher* modes most of the times. Once the searchers get their answers, they might want to share the results with other students or teachers. This is where the save-results utility provided by Buddy comes in handy.

2.2. Objectives Guiding Buddy

To summarize, the objectives guiding the development of Buddy are twofold:

1. **Dependability and Maintainability.** This is the most important objective. Connecting to source search engines should be easy. Buddy should be adaptable to changes in search engine sources. Additional search engines could easily be added into Buddy without recoding the data structures and methods. Any changes to the source search engines should require minimal modifications to the system, keeping the underlying data structures untouched. Modification should be done in a declarative approach. People maintaining the system need not be proficient in their programming skills.
2. **To be an appropriate Learning tool** (the name 'Buddy' reflects this objective). Buddy must cater to the needs of a searcher in any of the 4 searcher modes described in Section 2.1. Besides being able to search into multiple search engine databases, Buddy must be able to provide tools for disambiguation, such as query refinement and spelling suggestions so as to guide searchers in the novice mode. Buddy must also allow searchers (data gatherers and focused searchers) to search into specific directories and subjects. Buddy must enable use of Boolean expressions to make queries specific for searchers (focused searchers). Buddy must enable searchers (location searchers) to keep track of the searches they had done and to retrieve their previous search results. Finally, the system must minimize the number of sponsored links in the results.

3. SYSTEM DESIGN CONSIDERATIONS

Buddy was implemented using Java Development Kit (JDK) version 1.4.2 and Sun System Application Server 8. Java codes were written with EditPlus2 text editor. The web-based user interface (GUI) was implemented in JSP and OpenLaszlo (www.openlaszlo.org), the open-source platform for rich Internet applications. JSP files were written with Macromedia Dreamweaver 4. Some of the decisions and considerations in designing Buddy are:

- **Declarative Approach.** Properties of the data structures of source search engines are described in script files, contributing to maintainability. The integration of new search engine is simple.
- **Web scraping method.** There are a few open-source parsing tools [5][8] However, these are usually complex and incur a steep learning curve. Hence, we defined a simple web scraping method that uses Java Regular Expression and Pattern Matching API [6] and requires knowing only Regular expressions to modify the web scraping structure. E.g. 3 parts of the search result (shown in 3 different lines)

  "
  Yahoo!
  Welcome to Yahoo!, the world's most visited …"

  can be extracted using

(?:)	match URL
(.*?)(?:?)	match Title
(.*?)(?: <small><i>?)	match Description

  Expressions in bold define the groups of string that we scrape from the HTML page.
- **Multithreading.** Multithreading enables parallel request and retrieval of results from the parent search engines. Experiments have shown that parallel searches perform better than sequential searches [9].

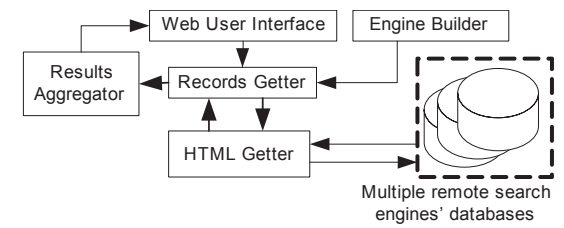
- **Query refinement method.** Most search engines use tools like WordNet [10] to help them perform query refinements. With the flexible web scraping method, Buddy scrapes query refinements and spelling suggestions from other search engines, as defined in configuration files. Hence, it is lightweight and useable on low-cost platforms.
- **Merging of results.** The simplest way of aggregation is to return all the results in one page without any post processing and re-ranking. This can lead to biased or overlapping results. In contrast, positional methods are computationally more efficient [11] and more precise. We use a positional method of ranking the merged results.
- **Web-based interface.** As the system was implemented using Java, Java Server Pages (JSP) was used to interact with the user and server. The GUI of the system aims to be intuitive and user-friendly. As the target users of the system are students and teachers, it is appropriate that the system is able to connect to subject-specific directories. Search engines are classified according to categories, namely Science, Math, Geography, History, Arts and General Search. E.g. Science category will include science-related search engines and directories – Scirus, Google's Science Directory and Yahoo's Science Directory. This will help searchers focus their search in a specific subject.

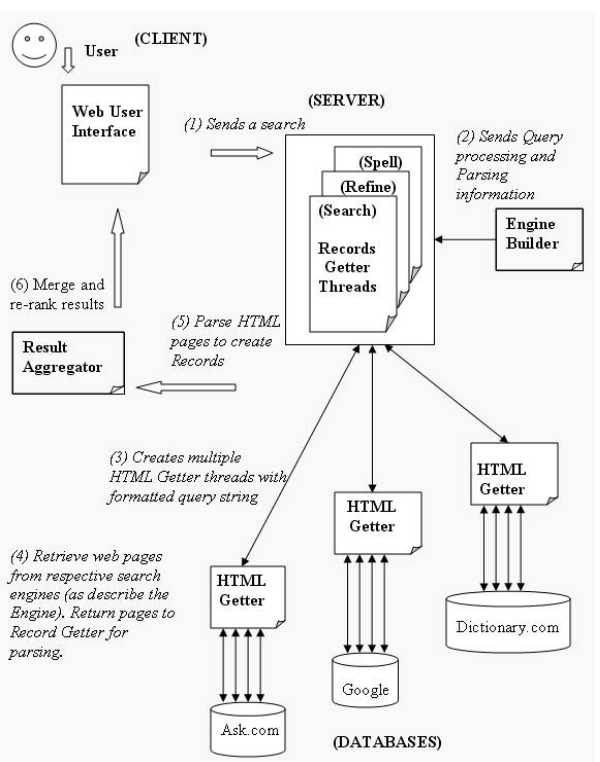
4. SYSTEM ARCHITECTURE

Buddy is made up of 5 components (see Figure 2):

1. **Web User Interface** – interacts with searchers
2. **Records Getter** – processes queries and parses/scrapes HTML pages; returns a vector of Records (results)
3. **HTML Getter** – retrieves HTML pages with the format query URL string; multithreading is used to speed up page retrieval.
4. **Engine Builder** – informs Records Getter to perform query processing and HTML parsing of different query/results format of different source search engines.
5. **Results aggregator** – merges, removes duplicates and re-ranks search records. Borda Positioning Rule was used to merge and re-rank the results, as it is relatively inexpensive, computationally efficient and has desirable properties such as anonymity, neutrality and consistency [11][12].

Figure 2. (Top) System architecture; (Bottom) Interaction among components





5. EXPERIMENTAL RESULTS

To test the performance of Buddy, 37 queries[3] were selected to obtain statistical results. These 37 search terms have been used previously in other studies [11] [12].

Experiments were done on a system with AMD Athlon XP 1600+ 1.4Ghz, 768 MB of RAM and a 2000 kbps Internet connection.

Two tests were conducted:

1. The first test connected the system to 6 search engines (Google, Yahoo, MSN Search, AllTheWeb, AltaVista, and Ask.com) individually to obtain 200 normal search results per query.
2. The second test meta-searched into 3 search engines (Google, Yahoo, and MSN Search) concurrently to return aggregated search results (only top 200 from each search engines is used; we would expect about 600 results per query). At the same time, we also searched for query refinements from Ask.com and Yahoo, and spelling suggestions from Dictionary.com.

The reader should note that we did not want to compare the performance of the various rank aggregation methods, nor compare performance with other meta-search engines. Instead, we wanted to evaluate the time taken to parse the HTML pages and aggregate the results against the total time taken to complete the task. This is to evaluate the amount of overhead (in terms of processing time) used for parsing and aggregating results. The tests were also to show that the system was running properly.

5.1. Test-1 Analysis

In this test, Buddy was used to search into 6 search engines individually to obtain 200 results per search engine per query.

Parsing Overhead

We are able to evaluate the parsing overhead incurred in this test. Parsing overhead is the amount of processing time required to parse HTML pages to create Vectors of Records.

From Figure 3, Buddy clocked an average of 2.5 seconds while searching into Google for 200 results per query. Ask.com took the longest time, with the bulk

Figure 3. Time taken for Buddy to search sequentially into each search engine

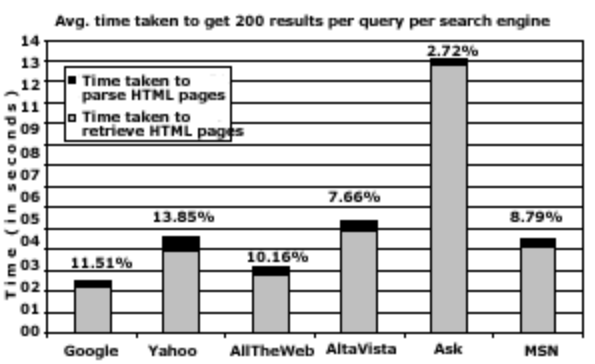


Table 1. Breakdown of records obtained by Buddy from each search engine (total 37 queries)

Search engine	Total no. of unique records	Total no. of duplicates	Total records
Google	7389	11	7400
Yahoo	7398	2	7400
AllTheWeb	7393	7	7400
Alta Vista	7395	1	7396
Ask	7357	11	7368
MSN Search	7312	48	7360
Total statistics	44244	80	44324

of the time spent on retrieving the HTML Page. We observed that on the average, only 7.24% of the processing time is involved in parsing. Bulk of the time is being used to retrieve the HTML pages instead.

Limitation Factor

We conclude that the performance of Buddy is limited by the connection speed to the search engines. We will expect the speed of combined search to be limited by the speed of slowest search engine selected. For instance, Searcher A selects Google and Ask.com. Searcher B selects Google and MSN Search. The system will take a longer time to obtain results for Searcher A because page retrieval from Ask.com is relatively the slowest (Figure 3).

Overlapping Records

It is interesting to note that there are actually a few (less than 1%) duplicates already present within the results of a search engine (Table 1). E.g. out of the 7400 records from Google, the system has removed 11 duplicates.

5.2. Test-2 Analysis

In Test Two, for every query, Buddy is used to perform the task of searching 3 search engines to obtain 600 results, to obtain query refinements and spelling suggestions. The final results are obtained by merging 200 results from each of the 3 search engines. We will expect some duplicated results to be removed. Here, we evaluated the overhead incurred for result aggregation. The top 3 fastest search engines from Test One (Google, Yahoo and MSN Search) were selected to participate in this test.

Results Aggregation Overhead and Performance

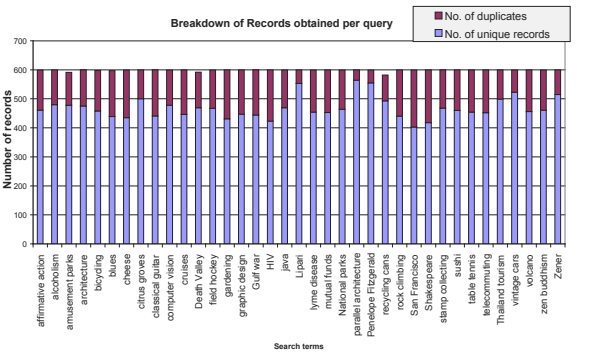
From Figure 4 (top), we can see that the average overhead cause by result aggregation is only 0.32%. The system has, on average, removed 21.78% of the total results that are overlapping. Also from Figure 4 (bottom), we can see that about one-fifth of the results are duplicates. For each query, the system takes about an average of 7.86 seconds to gather about 600 results from the 3 search engines, aggregates them and provides, on average, 90 query refinements. The performance is better than Helios [9], which took 12.4 seconds to retrieve 600 results.

In summary, the average time for Buddy to retrieve, parse and merge 600 results from Google, Yahoo and MSN Search is 7.8 seconds per query. This timing includes the retrieving and parsing of query refinement and spelling suggestions from Ask.com and Dictionary.com.

The performance of the system is greatly affected by the available bandwidth. Parsing and result aggregation overhead is not significant compared to that of HTML retrieving.

Figure 4. (Top) Statistics for Buddy to metasearch Google, Yahoo and MSN Search (200 results per search engine); (Bottom)Breakdown of records obtained per query

Search records from Google,Yahoo,MSN Search (200 results per search engine)							
	Query	No. of unique records	No. of duplicates	No. of Query refinements	No. of Spelling suggestions	Time Taken for Result Aggregations	Total Time taken (millisecs)
1	affirmative action	461	138	88	0	31	7438
2	alcoholism	480	120	115	0	15	12531
3	amusement parks	478	113	97	1	16	7531
4	architecture	475	125	140	0	31	6922
5	bicycling	458	140	65	0	32	7235
6	blues	439	158	138	0	16	7500
7	cheese	435	164	145	0	31	7171
8	citrus groves	500	100	5	7	32	7360
9	classical guitar	441	159	84	3	16	7656
10	computer vision	478	121	21	0	16	7657
11	cruises	446	154	133	0	46	7166
12	Death Valley	469	123	96	0	31	8438
13	field hockey	467	132	80	0	31	7687
14	gardening	431	169	149	0	31	7234
15	graphic design	447	153	132	0	15	6922
16	Gulf war	444	156	65	0	31	7109
17	HIV	423	177	103	0	32	11812
18	java	469	130	137	0	15	7469
19	Lipari	553	47	6	20	31	7781
20	lyme disease	454	145	78	0	15	7344
21	mutual funds	453	147	144	5	31	7843
22	National parks	464	135	144	2	32	8219
23	parallel architecture	564	36	11	0	15	7469
24	Penelope Fitzgerald	555	44	3	0	32	8172
25	recycling cans	493	89	48	1	15	7125
26	rock climbing	440	160	136	0	31	6750
27	San Francisco	403	196	135	0	16	7672
28	Shakespeare	418	182	138	0	16	7578
29	stamp collecting	468	132	60	0	31	11860
30	sushi	460	140	142	0	15	7203
31	table tennis	454	146	130	0	47	8500
32	telecommuting	452	148	58	0	16	7266
33	Thailand tourism	499	101	50	0	16	7813
34	vintage cars	523	76	58	6	32	7454
35	volcano	456	144	144	0	31	8094
36	zen buddhism	461	138	53	0	16	6797
37	Zener	515	85	26	49	16	7204
	Total Stats:	17326	4823	3357	94	922	290972
	% time spent on results aggregation=	Total Time Taken for Result Aggregations/Total Time taken =				922/290972 =	0.32%
	Avg. search time per query (millisecs) =	Total time taken/ 37 querys =				290972/37	7864.11
	% of records are duplicates=	Total number of duplicates/Total number of records=				4823/(17326+4823)	21.78%



6. CONCLUSIONS AND FUTURE WORK

Currently, Buddy can already connect and extract data from

1. 9 search engines (Google, Yahoo, MSN Search, AllTheWeb, AltaVista, Ask.com, Scirus, AOL and Lycos).
2. 15 Directories (Science, Math, Arts, History and Geography Directories) from Google, Yahoo and Open Directory Project.
3. Non-search engine sites such as Dictionary.com.

Web scraping has been an important method in the data extraction module of this system. It is an ad-hoc method that does not require us to learn extra APIs of the databases we want to connect to. It has enabled the system to extract data from virtually any free search engines that return results in HTML format.

It is interesting to note that the project has not involved external open source tools like XQuery or WordNet. The whole project has been done using standard Java

APIs. This shows the text processing power of Java language. Web scraping can be conveniently done using Java's Regular Expression and Patten Matching.

Declarative approach enables us to easily change the web scraping structure just by changing the regular expressions. We can also change the properties of the source search engines by editing the parameters in their descriptor script files. This does not require much programming knowledge to maintain the system.

This proposed meta-search system will be useful to searchers, especially to the target users in the education domain. The system is suitable for searchers who 1) want to have a wider range of answers to their queries from multiple sources 2) dread to see sponsored links 3) need help in query refinements and spelling suggestions 4) want to share their results with others or save their results for future reference. Thus, the system is certainly a suitable learning tool for students and teachers, and should find applicability in schools.

The system retrieves results straight from sources, without adding sponsored links to distract users. The system lets searchers have wider range of answers to their queries from multiple sources. The system is useful to novice searchers who need help in query refinements and spelling suggestions. The system also allows searchers to share their results with others or save their results for future reference. This system is certainly a suitable tool for learning for the students and teachers. Above all, it serves the primary objective of being a meta-search engine with increased dependability and maintainability.

Several problems were encountered and lessons learnt in the development of Buddy. The performance of the initial prototype was not desirable. This was because of the lack of parallelism being employed in the implementation. Retrieving HTML pages is usually the bottleneck of the whole search process because it takes a relatively long time to retrieve the pages. Subsequently, multithreading was used to retrieve the pages from the search engines, and the performance of the system is acceptable now. From the tests conducted (see Section 5), we can see that performance is greatly influenced by the amount of bandwidth available. The processing cost of parsing and result aggregations is not that high compared to that of retrieving HTML pages. Performance will be affected if the available connection speed is low. If we can speed up the HTML retrieval process by using faster Internet connection, the system's performance will improve.

Future work can include multi-language support, support for Really Simple Syndication (RSS) format [13] and classification of search results into appropriate categories.

REFERENCES

[1] J. Barker, "The Best Search Engines – UC Berkeley - Teaching Library Internet Workshops", *Finding Information on the Internet: A Tutorial*, UC Berkeley, 2006, Accessed 5 Sep 2006 from http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/SearchEngines.html

[2] G.R. Notess, "Little overlap despite database growth!", *Search Engine Statistics: Database Overlap*, Search Engine Showdown, Accessed 5 Sep 2006 from http://www.searchengineshowdown.com/statistics/overlap.shtml

[3] C. Sherman, "Metacrawlers and Metasearch Engines", *SearchEngineWatch*, 23 Mar 2005, Accessed 4 Sep 2006 from http://searchenginewatch.com/showPage.html?page=2156241

[4] PCWorld, "The Straight Story on Search Engines", *PCWorld Computing Center*, About.com, 2006, Accessed 5 Sep 2006 from http://pcworld.about.com/magazine/2007p115id97431.htm

[5] ceperez, "HTML Screen Scraping Tools Written in Java", *Manageability – Java Open Source*, Accessed 5 Sep 2006 from http://www.manageability.org/blog/stuff/screen-scraping-tools-written-in-java/view

[6] Sun Microsystems, "Pattern", *Sun Java2SEv1.4.2*, 2003, Accessed 5 Sep 2006 from http://java.sun.com/j2se/1.4.2/docs/api/java/util/regex/Pattern.html

[7] N.K. Agarwal and D.C.C. Poo, "Meeting knowledge management challenges through effective search", *Int. J. Business Information Systems*, 1(3), 2006, pp.292-309.

[8] B. Goetz, "Java theory and practice: Screen-scraping with Xquery", *Java technology / XML, IBM DeveloperWorks*, Accessed 4 Sep 2006 from http://www-128.ibm.com/developerworks/xml/library/j-jtp03225.html

[9] A. Gulli and A. Signorini, "Building an open source meta-search engine", Special interest tracks and posters of 14th WWW Conf., May 10-14, 2005, Chiba, Japan.

[10] Princeton University, *WordNet: a lexical database for the English language*, Cognitive Science Laboratory, Accessed 3 Sep 2006 from http://wordnet.princeton.edu/

[11] M.S. Mahabhashyam and P. Singitham, "Tadpole: A Meta search engine Evaluation of Meta Search ranking strategies", *CS276A Project, Stanford University*, Fall 2002, Accessed 7 Sep 2006 from http://www.stanford.edu/class/cs276a/projects/reports/mmahathi-pavan.doc

[12] C. Dwork, R. Kumar, M. Noar and D. Sivakumar, "Rank aggregation methods for the web" In *Proceedings of the 10th International Conf. on the World Wide Web (WWW10)*, May 1-5, 2001, Hong Kong, ACM Press and Addison Wesley, pp.613-622.

[13] D. Winer, *RSS 2.0 Specification*, Berkman Center for Internet & Society, Harvard Law School, Accessed 4 Sep 2006 from http://blogs.law.harvard.edu/tech/rss

ENDNOTES

1 There are many misspelt words that Google cannot detect e.g. arrowplane, arrowdynamic, brase, buule, colar, canntin, diform, doubl, etc.

2 'Novice' was termed 'learner' and 'location searcher' was termed 'location seeker' in [7]. The terms have subsequently been revised to remove ambiguity.

3 affirmative action, alcoholism, amusement parks, architecture, bicycling, blues, cheese, citrus groves, classical guitar, computer vision, cruises, Death Valley, field hockey, gardening, graphic design, Gulf war, HIV, java, Lipari, lyme disease, mutual funds, National parks, parallel architecture, Penelope Fitzgerald, recycling cans, rock climbing, San Francisco, Shakespeare, stamp collecting, sushi, table tennis, telecommuting, Thailand tourism, vintage cars, volcano, zen buddhism, and Zener.