The MetaCrawler Architecture for Resource Aggregation on the Web

Erik Selberg   Oren Etzioni

Department of Computer Science and Engineering

University of Washington

Seattle, WA 98195

{selberg, etzioni}@cs.washington.edu

http://www.cs.washington.edu/homes/{selberg, etzioni}

http://www.cs.washington.edu/research/metacrawler

November 8, 1996

1   Introduction

The MetaCrawler Softbot is a parallel Web search service that has been available at the University of Washington since June of 1995. It provides users with a single interface with which they can query popular general-purpose Web search services, such as Lycos[6] and AltaVista[1], and has some sophisticated features that allow it to obtain results of much higher quality than simply regurgitating the output from each search service. In this article, we briefly outline the motivation for MetaCrawler and highlight previous work, and then discuss the architecture of MetaCrawler and how it enables MetaCrawler to perform well and to scale and adapt to a dynamic Internet.

1.1   Motivation

Since its inception, the World Wide Web has grown at a staggering rate. As a result, finding information on the Web is often difficult. To address this problem, search services such as WebCrawler[8] and Lycos [7] were created which use robots to collect information and index it at one place, and directory services such as Yahoo were created using human editors to organize and categorize Web pages. Both types of services make it easier for users to find the information they need. These services are also very popular as people like going to one place and discovering the information they want. Several other services have also recently emerged, such as AltaVista[1] and HotBot[5], which are attempting to form the most up-to-date and complete index of the Web.

A common belief is that the majority of Web search services are roughly the same. They all index the same content, and all use reasonable Information Retrieval (IR) techniques to map a query to potentially relevant URLs. Our experiments suggest otherwise.

1.2 Previous Findings

We found that each service returns different documents for the same query. Also, there is an inherent time lag in each services' results – one service may have an index of a document that is only a day or two old, whereas another may have an index that is a month old. Thus, you never know if the references returned are fresh references, or stale references that were once appropriate. A final complication is that each service uses a different ranking algorithm. One service might rank pages that are highly relevant to a given query in the top positions, whereas another may not. In practice, each ranking algorithm tends to work well for certain classes of queries, but performs poorly in others. Taken all together, each service has an extremely disparate set of documents in their index, not only in terms of which documents are indexed, but also in terms of *when* each document was indexed. We previously demonstrated that by relying exclusively on a single search engine instead of the MetaCrawler, users could miss over 77% of the references they would find most relevant[10].

We have also observed that the interface of each search service is often not optimal. Many of the features users desire, such as phrase searching or filtering by location, are often absent or require a syntax difficult for average users. Furthermore, many of the references returned by existing services were unavailable. Earlier experiments demonstrated that by examining pages to ensure they exist and have quality content, as well as by providing the user with added expressive power, over 75% of the references returned to the user could be automatically determined to be irrelevant and subsequently removed[10].

2 The MetaCrawler Softbot

The MetaCrawler Softbot was created to address the problems outlined above. MetaCrawler is a *Soft*ware ro*bot* that aggregates Web search services for users. MetaCrawler presents users with a single unified interface. Users enter queries, and MetaCrawler forwards those queries in parallel to the sundry search services. MetaCrawler then collates the results and ranks them into a single list, returning to the user the sum of knowledge from the best Web search services. The key idea is that the MetaCrawler allows the user to express *what* to search for and frees the user from having to remember *where* or *how*.

MetaCrawler has several features that enable the user to obtain high quality results. Users can instruct the MetaCrawler to find pages with either *all* of the words in their query, *any* of the words in their query, or all of the words in their query as a *phrase*. Users can also specify if a word must or must not appear in any reference returned, as well as if a reference comes from a particular geographic or logical location (e.g. France or `boeing.com`). MetaCrawler has several additional features that will not be discussed here, but are explained in detail at the MetaCrawler WWW site[9].

In order to accomplish its goal, MetaCrawler must be adept at several tasks. It needs to take user queries and format them appropriately for each search service. Next, it must be able to correctly parse the results and aggregate the results from the other services. Finally, it must be able to analyze the results to eliminate duplicates and perform other checks to ensure quality. Figure 1 details the MetaCrawler's control flow.

It is important to note that accomplishing these tasks is only sufficient to get MetaCrawler to work. There are several additional requirements in order to make it usable, which in turn strongly impact how MetaCrawler is designed. First and foremost, MetaCrawler needs a user interface that the average person can use. Second, MetaCrawler must be able to perform its tasks for the user as quickly as possible. Finally, MetaCrawler must be able to adapt to a rapidly changing environment.
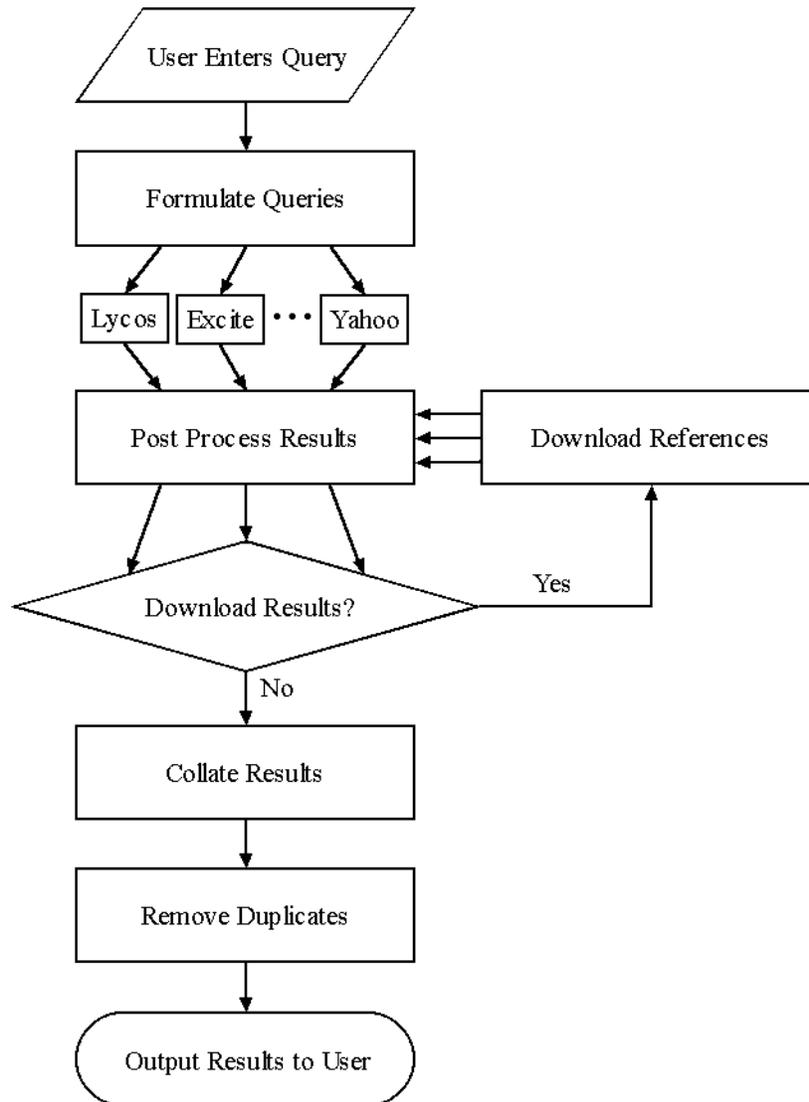
Figure 1: MetaCrawler Control Flow

2.1 Understanding Query and Output Formats

MetaCrawler has to understand both the query format and feature set supported by each service. Understanding the query format is not difficult, but there are some details involved. For example, Alta Vista prefers searching for phrases by enclosing the phrase in double quotes, e.g. "Erik Selberg" whereas HotBot uses a menu option that selects phrase searching.

Translating a user query into the proper format for each search service is only half of the equation. MetaCrawler must also parse the results of the query and collate results from different services into a single, coherent list. This task is difficult because each service uses a different ranking criterion, and some services don't report any information about the ranking besides the order. Furthermore, because each service's robot is slightly different, the URLs recorded for identical pages are often different; for example, one robot may find `http://www.cs.washington.edu/homes/speed/home.html` while another may find `http://www.cs.washington.edu/homes/selberg`, both of which point to the same document.

2.2 Post-Processing References

Supporting a rich feature set can be problematic. Some services have all the features that MetaCrawler uses, whereas others are lacking in particular areas such as phrase searching or weighing words differently. MetaCrawler implements these and other features not available from any service by downloading the pages referred by each service and analyzing them directly. The time required to download all pages in parallel is reasonable, usually adding no more than a couple of minutes to the search time. In addition, MetaCrawler is able to process results that have arrived while waiting for others. Therefore, minimal computation time required after all the references have been retrieved. It is important to note that while this does improve quality substantially, it is a user-enabled option[1] as most users prefer to have the results displayed quickly.

Downloading and analyzing references locally has proven a powerful tool. We can implement features not found in some services. For example, Lycos does not currently handle phrase searching. MetaCrawler simulates phrase searching by sending Lycos a query for documents containing all the query words, downloads the resulting pages, and extracts those pages that actually contain the appropriate phrase. Using similar methods, we are able to handle other features commonly found in some, but not all, search services, such as requiring words to be either present or absent in pages.

In addition to simulating features on a particular search service, we are able to implement new features. The most obvious is that by downloading a reference in real time, we have verified that it exists. This has proven to be a popular feature in itself. We are also able to analyze the pages for sophisticated ranking and clustering algorithms, and are able to extract words from the documents that may be useful for refining the query.

2.3 Collation and Duplicate Elimination

Detecting duplicates references is difficult without the full contents of a particular page, due to host name aliases, symbolic links, redirections, and other forms of obfuscation. To eliminate duplicates, MetaCrawler uses a sophisticated comparison algorithm. When comparing two URLs, it first checks that the domains are the same (i.e. is `www.cs.washington.edu` really `bauhaus.cs.washington.edu`?). It then checks if the paths are the same using standard aliases (e.g. a URL of the form `http://www.some.dom/foo` often refers

---

[1] The post-processing option is currently disabled pending additional coding that will ensure MetaCrawler does not cause an undue load on remote sites by downloading pages.

to `http://www.some.dom/foo/index.html`). Finally, if the domains are the same and the paths aren't, it looks at the titles of the pages. If they are the same, it assumes that they are aliases, although rather than deleting the reference it puts the link beneath the original. MetaCrawler is able to make an even more concrete determination if it can download the page and compare the full text. There are issues that still need to be resolved, such as mirrored pages, but our algorithm has worked well enough in practice.

MetaCrawler uses a *confidence score* to determine how close a reference matches the query. A higher confidence score indicates a more relevant document. To calculate each references' confidence score, MetaCrawler first distributes the confidence scores returned by each service into the range 0 to 1000. Thus, the top pick from each service will have a confidence score of 1000. Then, the MetaCrawler eliminates duplicates, and adds to the duplicated reference's score the sum of the removed references confidence scores. In essence, this allows services to vote for the best reference, as a reference returned by three services will most likely have a higher total than a reference returned by one. In addition, we provide the option of seeing the results sorted by location rather than relevance.

2.4   User Interface

MetaCrawler must have a reasonable user interface if the average Web user is to interact with it. Visually, there is a single text entry box, a set of radio buttons which describe the logic of the query (e.g. "All these words," "As a phrase"), various options describing what results are desirable (e.g. results from the user's country, .edu sites, etc), and sundry performance metrics (e.g. how long to wait). This preserves the metaphor which people prefer and understand – one single interface with which to find information on the Web. It is important that we do not assume that the user is skilled in Boolean logic, as a large portion of the feedback we received from users suggested that they often confused Boolean expressions (using AND when they meant OR, believing that OR meant exclusive OR, etc.).

2.5   Performance

In order for MetaCrawler to be practical for mass use, it must be fast. A naive implementation would wait until all search services returned results and then wait again until each reference has been downloaded. This creates a barrier. This problem compounds during peak rates, when the slowest services can take on the order of a minute longer to return results than the faster services.

We have addressed the performance issues using several methods. First, we have installed user-modifiable timeouts. This way, if some services are overly slow, MetaCrawler will time them out relatively quickly. By default, this time is 30 seconds. Second, we only download references when needed, i.e. when the user activates a particular option, such as phrase searching. As search services become more sophisticated, we can implement more features without downloading references. We have also experimented with an explicit switch to allow users to control when MetaCrawler does or does not download references, although initial user feedback has indicated that users prefer implicit control versus dealing with yet another option.

We are also putting substantial effort into showing *something* to the user as quickly as possible. While this does not actually improve the speed of MetaCrawler, showing something quickly gives the perception of increased speed, and gives feedback as to how long the query will take. MetaCrawler displays when a service returns and how many results it returned. Our new Java interface prototype lets users see and explore the partial results. This allows the user to examine what has come in quickly, possibly by following some references. If the desired reference is present, then the user can quickly move on. Otherwise, no time is lost; the query is still running, and has probably completed by the time the user is finished perusing the initial
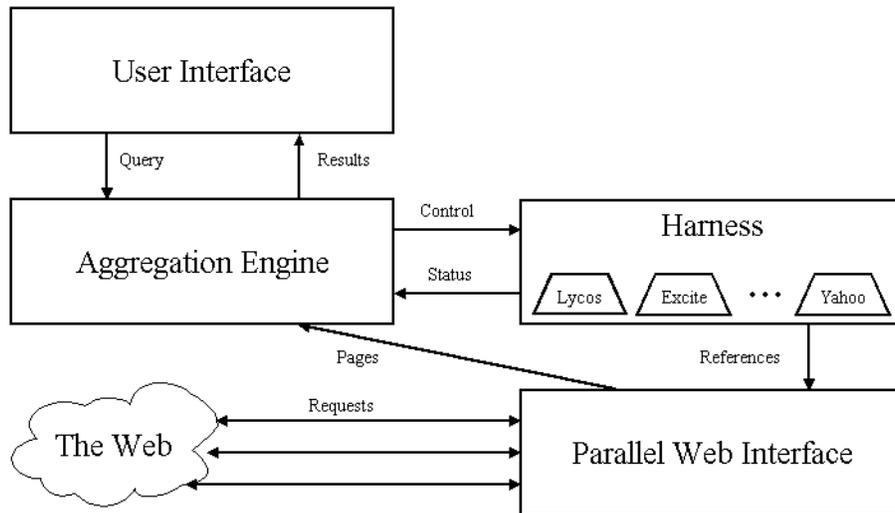
Figure 2: MetaCrawler Architecture

results. Early feedback from users on this feature has been positive.

3   Architectural Design

MetaCrawler is designed in a modular fashion, as depicted in Figure 2. The main components are the User Interface, the Aggregation Engine, the Parallel Web Interface, and the Harness. The User Interface is simply the layer that translates user queries and options into the appropriate parameters. These are then sent to the Aggregation Engine, which is responsible for obtaining the initial references from each service, post-processing the references, eliminating duplicates, and collating and outputting the results back to the User Interface for proper display. Most of the sophisticated features of MetaCrawler reside in this component. The Parallel Web Interface is responsible for downloading HTML pages from the Web, including sending queries and obtaining results from each search service.

The Harness is the crux of the design; it's where the service-specific information is kept. The Harness receives control information detailing which references to obtain; it then formats the request properly and sends the reference to the Parallel Web Interface, which then returns a page to the Aggregation Engine. It also sends some status information back to the Aggregation Engine that is used to measure progress. The Harness is implemented as a collection of modules, where each module represents a particular service. It is designed so that modules can be added, modified, and removed without impacting the rest of MetaCrawler.

MetaCrawler's architecture provides several advantages. It provides a layer of abstraction above traditional search services, which allows for greater adaptability, scalability, and portability as the Web grows and changes.

3.1 Adaptability

Search services are extremely volatile in today's Internet. New search services are being launched continually. Almost as frequently is the rate at which search services upgrade their systems, which often means that their searching interface changes. Finally, while many search services are emerging, there are also a number of services being removed or moved to new sites, leaving and dangling references elsewhere in the Web. MetaCrawler's modular design allows for new services to be added, modified, and removed quickly, thus allowing it to easily adapt to a changing Internet.

3.2 Portability

The MetaCrawler Softbot is an intelligent interface to powerful remote services. It does not require large databases or large amounts of memory. This provides great flexibility in its location. Currently, MetaCrawler exists as a universally accessible service at the University of Washington. However, we have created prototype implementations that reside on the user's machine. We have also had great success in compiling MetaCrawler on different architectures, including DEC OSF, Linux, and even Windows. The MetaCrawler Softbot can run on most machines currently available with minimal effort, which means that most users can use MetaCrawler, either locally or remotely, without needing to invest in expensive hardware. Further, MetaCrawler is not rooted to any platform-specific technology, so as machines improve MetaCrawler will be able to adapt and operate on those new machines as well.

3.3 Scalability

As the Internet grows, so does its user base. It is important that the MetaCrawler scale to handle these users. The load on the servers handling the MetaCrawler service scales linearly with the number of requests per day. Unfortunately, given the expected usage projections of the Web, even linear scaling may require a daunting investment in hardware. However, using implementations resident on users' machines, we are able to scale without the need to add more machines. Every user is responsible for their own network and computational requirements, and they only need contact the server periodically to obtain updated service lists and other new information.

4 Future Work

There are several promising areas for future work with the MetaCrawler Softbot. The first is to have MetaCrawler scale with the number of search services; currently, it accesses every service on each query. While that is acceptable for a limited number of search services, a broadcast protocol won't work with a large number of small or special purpose search services. One possibility is to use a learning-based approach similar to SavvySearch[3], where MetaCrawler learns which services to use based on prior experience. Another method is to take an approach similar to GlOSS[4] or the Content Router[11] which use information obtained directly from the search services to route queries appropriately.

Other areas of research involve improving MetaCrawler's output method; currently, we support ranking by relevance and location. Other options are being explored, such as clustering methods [2].

5 Conclusion

MetaCrawler presents users with a single intelligent interface that controls several powerful information sources, formats the user's queries for each service it uses, collects the references obtained from those services, and optionally downloads those references to ensure availability and quality. It then removes duplicate references and collates the rest into a single list for the user. The user need know only *what* he or she is looking for the MetaCrawler Softbot takes care of *how* and *where*. We have paid special attention to performance, making it a practical tool for Web searching.

MetaCrawler's architecture is a modular harness; we are able to plug in, modify, and remove services easily. This enables us to combine many different resources without difficulty. It is adaptable to new and modified services. Because of its minimal resource requirements, it is very portable, able to exist as a server on enterprise-scale UNIX servers or as a user application on a Windows box. Also, it exhibits reasonable scaling properties, especially when viewed as a client application. The research service has been available to the public for over a year, with a current query rate of over 150,000 per day.

6 Acknowledgments

MetaCrawler would not exist were it not for the help of a great many people. Greg Lauckhart has been invaluable in its recent development, in particular the development of the Java interface and numerous internal architecture improvements. Nancy Johnson Burr, Voradesh Yenbut, Rick White, Art Dong, Steve Corbato, Tom Profit, Terry Gray, and Erik Lundberg have been instrumental in keeping the MetaCrawler service up and running. Thanks also to Ed Lazowska for his insight and support. Finally, thanks to Mary Kaye Rodgers for her patience with long nights and impending deadlines.

7 About the Authors

Erik Selberg, *selberg@cs.washington.edu*, http://www.cs.washington.edu/homes/selberg
Department of Computer Science and Engineering
Box 352350
University of Washington
Seattle, WA 98195

Erik Selberg earned his bachelor's degree from Carnegie Mellon University in 1993 with a double major in computer science and logic, receiving the first Allen Newell Award for Excellence in Undergraduate Research. He earned his master's degree from the University of Washington in 1995 and is continuing his study towards his Ph.D.

In 1995, he created the MetaCrawler, a parallel Web search service. The MetaCrawler has received numerous awards and mentions, including being chosen as one of three finalists in the C|Net Awards for Best Internet Search Engine and reviews in Forbes, Business Week, and IEEE Computer.

He is primarily interested in Artificial Intelligence and Information Retrieval, especially as it pertains to the World Wide Web. He also has interests regarding system performance, security, and multi-agent coordination and planning. He also enjoys skiing, raquetball, and 5-star Thai cuisine.

Oren Etzioni, *etzioni@cs.washington.edu*, http://www.cs.washington.edu/homes/etzioni
Department of Computer Science and Engineering

Box 352350
University of Washington
Seattle, WA 98195

Oren Etzioni, Associate Professor, received his bachelor's degree in computer science from Harvard University in June 1986, and his Ph.D. from Carnegie Mellon University in January 1991. He joined the University of Washington as assistant professor of computer science and engineering in February 1991, and was promoted to Associate Professor in 1996.

In the fall of 1991, he launched the Internet Softbot project. In 1993, Etzioni received an NSF Young Investigator Award. In 1995, the Internet Softbot was chosen as one of 5 finalists in the National DISCOVER Awards for Technological Innovation in Computer Software. In 1996, MetaCrawler was chosen as one of 3 finalists in the C|NET Awards for Best Internet Search Engine. His group's work on software agents has been featured in The Economist, Business Week, Discover Magazine, Forbes Magazine, and The New Scientist.

His research interests include: Software Agents, Web Navigation and Search Technology, and Human-Computer Interaction. He used to have hobbies, but he is now a proud father instead.

References

[1] Digital Equipment Corporation. AltaVista Home Page.
URL: http://www.altavista.digital.com.

[2] Douglas R. Cutting, Jan 0. Pedersen, David Karger, and John W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proceedings of the 1992 ACM SIGIR Conference*, Copenhagen, Denmark, June 1992.
URL: http://corp.excite.com/people/cutting/papers/sigir92.ps.

[3] Daniel Dreilinger. Integrating Heterogeneous WWW Search Engines. Master's thesis, Colorado State University, May 1995.

[4] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The Effectiveness of GlOSS for the Text Database Discovery Problem. In *Proceedings of the 1994 ACM SIGMOD Conference*, Minneapolis, MN, May 1994.
URL: ftp://db.stanford.edu/pub/gravano/1994/stan.cs.tn.93.002.sigmod94.ps.

[5] Inktomi, Inc. HotBot Home Page.
URL: http://www.hotbot.com.

[6] Michael Mauldin. Lycos Home Page.
URL: http://lycos.cs.cmu.edu.

[7] Michael L. Mauldin and John R. R. Leavitt. Web Agent Related Research at the Center for Machine Translation. In *Proceedings of SIGNIDR V*, McLean, Virginia, August 1994.

[8] Brian Pinkerton. Finding What People Want: Experiences with the WebCrawler. In *Proc. 2nd World Wide Web Conference*, Chicago, IL USA, October 1994.

[9] Erik Selberg and Oren Etzioni. MetaCrawler Home Page.
URL: http://www.cs.washington.edu/research/metacrawler.

[10] Erik Selberg and Oren Etzioni. Multi-Service Search and Comparison Using the MetaCrawler. In *Proc. 4th World Wide Web Conference*, Boston, MA USA, December 1995.
URL: http://metacrawler.cs.washington.edu:8080/papers/www4/html/Overview.html.

[11] Mark A. Sheldon, Andrzej Duda, Ron Weiss, and David K. Gifford. Discover: A Resource Discovery System based on Content Routing. In *Proc. 3rd World Wide Web Conference*, Elsevier, North Holland, April 1995.
URL: http://www-psrg.lcs.mit.edu/ftpdir/papers/www95.ps.