

Building an Open Source Meta Search Engine

Antonio Gulli
Università di Pisa, Informatica
gulli@di.unipi.it

Alessio Signorini
University of Iowa, Computer Science
alessio-signorini@uiowa.edu

ABSTRACT

In this short paper we introduce Helios, a flexible and efficient open source meta-search engine. Helios currently runs on the top of 16 search engines (in Web, Books, News, and Academic publication domains), but additional ones can be easily plugged in. Furthermore, we report some performance results obtained during its development.

1. INTRODUCTION

A recent study [9] estimated the size of publicly indexable web at more than 11.5 billion pages. Beside, the index intersection between the largest available search engines – namely Google, Yahoo!, MSN, Ask/Teoma – is estimated to be less than the 28.8%. A study [1] showed that the 44% of searchers regularly use only a single search engine, the 48% use just two or three search engines, and only the 7% use more than three. Another study conducted by Jux2 [2] pointed out that Google and Yahoo! share only 3.8 of their top 10 results, among the 500 most popular search terms. In a separate test of 91 random searches, they also found that Google and Yahoo! share only 23% of their top 100 results. They claim that “If the search engines are providing top results that are very different from each other, then by using only one search engine, Internet searchers are potentially missing relevant results”.

As a consequence, meta search engines are relevant for many reasons. For instance, they allow to: (i) integrate answers provided by different search engines (ii) compare rank positions (iii) provide advanced search features on the top of commodity search engines (e.g. clustering, question answering and personalized results).

There are many industrial meta-search engines: Vivisimo and Dogpile are commercial clustering engines that group results drawn on-the-fly from other remote search engines. Jux2 is an industrial meta-search engine that compares on three search engines the different rank positions assumed by a set of URLs. A complete list of meta-search engines is available at [3].

In the academic literature, there are many proposals for meta-searching. [10] proposed to work by downloading and analyzing the individual documents, rather than working with the list of summaries returned by search engines. This approach can have evident performance problems. [11] is a good survey of techniques that have been proposed to tackle several underlying challenges in building a good meta-search engine. [6] proposed system and method for improving answer relevance in meta-search engines. [12, 13, 7] discussed several strategies for combining the ranked results returned from multiple search engines.

Our contribution: In this short paper we introduce Helios, a com-

plete meta-search engine for retrieving, parsing, combining, and reporting results provided by many search engines. Our contributes are the followings:

- Helios is a full working open-source meta search engine available at <http://www.cs.uiowa.edu/~assignori/helios/>. Different research groups can use the system to interact with many search engines and develop their service on the top of them¹. Helios is currently used by a number of academic research projects such as a personalized web-snippets clustering engine [8], a rank comparison engine [4], and an experiment for measuring the size of the Web [9];
- Helios can extract results from different domains. Currently it supports a set of 15 search engine on Web, News, Books, and Academic Publications domain (A9, About, AllTheWeb, Altavista, AOL Search, eSpotting, FindWhat, Gigablast, Google, LookSmart, Mozdex, Msn, Overture, Ask/Teoma, and Yahoo!). Moreover, it is easy to plug-in a new engine;
- Helios was intensively engineered to be efficient, light-weight, and then usable also on low cost platforms; The experimental results are a benchmark we offer to the research community.

2. HELIOS ARCHITECTURE

In this section we describe the architecture of Helios (Figure 1).

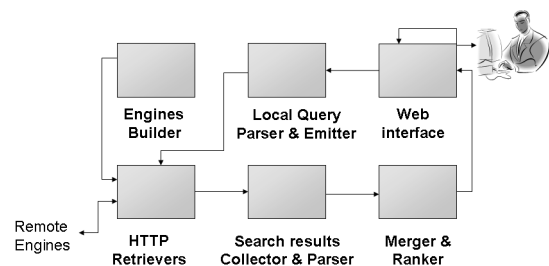


Figure 1: Architecture of Helios.

The *Web Interface* allows users to submit their queries and select the desired search engines among those configured in the system. This information is interpreted by the *Local Query Parser & Emitter* that re-writes queries in a format appropriate for each chosen engine. The *Engines Builder* maintains all the settings necessary to communicate with the remote search engines. *HTTP Retrievers* modules handle the network communications. As soon as search results are available, the *Search Results Collector & Parser* extracts the relevant information, and format it in XML. This choice was crucial to use Helios in the heterogeneous set of academic

¹provided that they don't violate any licence of use

projects previously described. Users can adopt the standard *Merger & Ranker* module or integrate their own customized one.

For efficiency reasons, Helios maintains parallel TCP connections with remote search engines. The communication model adopted is async I/O [14]. This is useful for many reasons: (i) the system is not overloaded with hundreds of threads; (ii) the connection cost is reduced to few μsec , since parallel connections allow to retrieve data from one server while starting the connection to a second one, sending data to a third one, and so on; (iii) for a given query, it is possible to retrieve in parallel and from the same search engine many different result pages.

The integration of a new engine is simple. A configuration file allows to specify many parameters (e.g. query re-writing rules, ip address, parsing rules and so on). We defined a simple but efficient parsing language which allows to search strings, move a cursor over an input string, extract substring, delete or rewrite them. The language provides some programming constructs such as *if*, *until*, and *jump*. The language does not provide the same power of regular expressions but can process the retrieved search results in a fast efficient way (see Figure 4).

3. EXPERIMENTAL RESULTS

Experiments were conducted on a dual Pentium IV 2.60Ghz, with 1.5Gb of RAM memory and a 100Mbps internet connection. In all our experiments the resources usage was negligible. For space reasons, we report a subset of our experiments.

Parallel searches on multiple engines: We used Altavista, Gigablast, Google, Looksmart, Ask/Teoma and Yahoo! as test bed engines. Downloading sequentially the top 100 results from each search engine in the test bed – a total of 600 search results – required 12.4 seconds. This test was done using *wget* [5], a common http retriever tool. Helios, we can retrieve in parallel and parse the same results in less than 4.6 seconds². Note this time was largely dominated by Ask/Teoma which gives a maximum of 15 results per page. In this case, Helios was configured to request 7 pages in parallel to obtain the requested top 100 results.

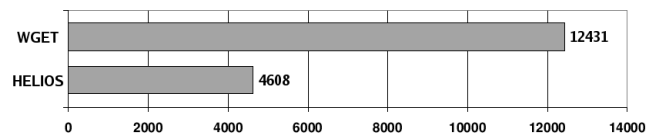


Figure 2: helios vs wget. time is in seconds.

Parallel searches on one engine: Another test on Ask/Teoma showed that a sequence of 7 serial interrogations required about 5.7 seconds to be completed. Exploiting parallel requests, the time was reduced to less than 1 second.³

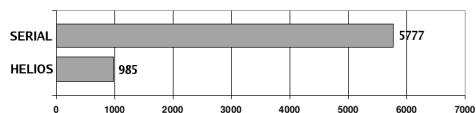


Figure 3: Downloading times of 7 pages from Ask/Teoma

Parsing time: For the query “god”, Helios parsed 100 results drawn from Altavista, Google and Yahoo! in less than 16 milliseconds.

²quale query

³query?

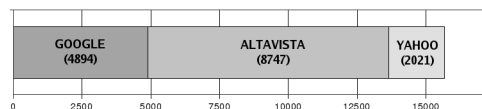


Figure 4: Parsing performance in μsec for the query “god”, and top 100-results

Overall performances: Helios required less than 20 seconds to retrieve and parse 3000 results (10 pages of 100 results for each search engine) from Altavista, Google and Yahoo! for the query “flowers”.

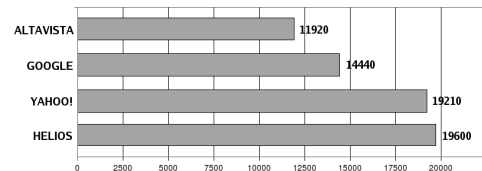


Figure 5: Retrieving+parsing times in μsec for 3000 urls for the query “flowers”.

Retrieving and parsing the top 200 results from Altavista, Google and Yahoo! – a total of 600 results – required only 2.21 seconds, using 6 parallel connections. Google required 2.39 seconds to return its top 600 results using 6 parallel connections.⁴

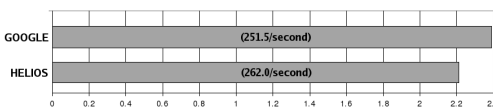


Figure 6: Comparison of retrieving times between Google and Helios for the query “banana”

Processing the top 100-results from (A9, About, AllTheWeb, Altavista, AOL Search, eSpotting, FindWhat, Gigablast, Google, LookSmart, Mozdex, Msn, Overture, Ask/Teoma, Yahoo!) – a total of 1355 results – required less than 9 seconds to be completed with a 3% of average CPU usage.⁵

4. REFERENCES

- [1] http://www.pewinternet.org/pdfs/PIP_Searchengine_users.pdf.
- [2] <http://www.jux2.com/stats.php>.
- [3] <http://searchenginewatch.com/links/article.php/2156241>.
- [4] <http://rankcomparison.di.unipi.it/>.
- [5] <http://www.gnu.org/software/wget/>.
- [6] Chidlovskii. System and method for improving answer relevance in meta-search engines. U.S. Pat. 6829599, 2004.
- [7] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee. Comparing and aggregating rankings with ties. In *PODS*, 2004.
- [8] P. Ferragina and A. Gulli. The anatomy of a clustering engine for web, books, news snippets. In *ICDM*, 2004.
- [9] A. Gulli and A. Signorini. Web is more than 11.5 billion pages. submitted for publication, 2005.
- [10] S. Lawrence and C. L. Giles. Inquirus, the NECI meta search engine. In *WWW7*, 1998.
- [11] W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. In *ACM Computing Surveys*, 2002.
- [12] M. E. Renda and U. Straccia. Web metasearch: Rank vs. score based rank aggregation methods. In *SAC*, 2003.
- [13] F. Gibb S. Wu, F. Crestani. New methods of results merging for distributed information retrieval. In *Distributed Multimedia Information Retrieval*, 2003.

⁴dire sempre la query non solo nella caption

⁵quale query?, figura? come si differenzia da prima.

[14] Richard Stevens. *UNIX Network Programming*, volume 2. Prentice Hall, 1999.