Building an Open Source Meta-Search Engine

A. Gulli
Dipartimento di Informatica, University of Pisa
gulli@di.unipi.it

A. 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 18 search engines (in Web, Books, News, and Academic publication domains), but additional search engines can be easily plugged in. We also report some performance mesured during its development.*

Categories and Subject Descriptors
H.3.3 [**Information Storage And Retrieval**]: Information Search and Retrieval

General Terms
Design, Experimentation, Measurement

Keywords
Meta Search Engines, Open Source

1. INTRODUCTION

A recent study [8] estimated the size of publicly indexable web at more than 11.5 billion pages. Furthermore, the index intersection between the largest available search engines – namely Google, Yahoo!, MSN, Ask/Teoma – is estimated to be 28.8%. A study [1] showed that 44% of searchers regularly use only a single search engine, 48% use just two or three search engines, and only 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 useful for many reasons. For instance, they allow (i) integration of search results provided by different engines, (ii) comparison of rank positions, (iii) advanced search features on top of commodity engines (e.g. Clustering, QA 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 list of meta-search engines is in [3].

In the academic literature, there are many proposals for meta-searching. [9] proposes to work by downloading the individual documents, rather than working with the list of snippets returned by search engines. This approach has evident performance problems. [10] reports a survey of techniques that have been proposed to tackle several underlying challenges in building a meta-search engine. [5] discusses methods for improving answer relevance in meta-search engines. [11, 12, 6] propose several strategies for combining the ranked results returned from multiple search engines.

Our contribution: In this short paper we introduce Helios, a complete meta-search engine for retrieving, parsing, merging, and reporting results provided by many search engines. Our contributions are the followings:
(1) Helios is a full working open-source meta-search engine available at *http://www.cs.uiowa.edu/∼asignori/helios/*. Different research groups can use the system to interact with many engines and develop their services on the top of them (provided that they don't violate any licence of use). Helios is currently used by a number of academic research projects such as a personalized web-snippets clustering engine [7], a rank comparison engine [4], and an experiment for measuring the size of the Web [8];
(2) Helios supports a set of 18 engines on Web, News, Books, and Academic Publications domain (A9, About, AllTheWeb, Altavista, AOL Search, eSpotting, FindWhat, Gigablast, Google, LookSmart, Mozdex, Msn, Overture, Ask/Teoma, Yahoo!, Google News, Google Scholar, and Yahoo News). Moreover, it is easy to plug-in a new engine;
(3) Helios was intensively engineered to be efficient, lightweight, and thus usable on low cost platforms. The experimental results obtained are a benchmark for the community.

2. HELIOS ARCHITECTURE

In this section we describe the architecture of Helios (Fig. 1). The *Web Interface* allows users to submit their queries and to select the desired search engines among those supported by the system. This information is interpreted by the *Local Query Parser & Emitter* that re-writes queries in the appropriate format for the chosen engines. The *Engines Builder* maintains all the settings necessary to communicate with the remote search engines. The *HTTP Retrievers* modules handle the network communications. As soon as search results are available, the *Search Results Collector & Parser* extracts the relevant information, and returns it using XML. This choice allowed Helios to be easily used in the heterogeneous set of academic projects previously described. Users can adopt the standard *Merger & Ranker* module for search results or integrate their customized one.

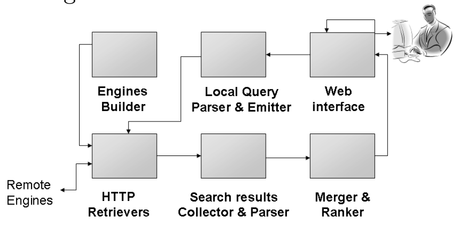

Figure 1: Architecture of Helios.

To achieve its high-performance, Helios utilizes async I/O [13] and parallel TCP connections, with the remote search engines. This is useful for two reasons: (i) the system is not overloaded with hundreds of threads; (ii) the connection cost is reduced to a 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. We remark that for a given query, it is possible to exploit both parallelism among different search engines and within a single engine.

The integration of a new engine is simple: a configuration file is used to specify the engine parameters (e.g. query re-writing rules, ip address, parsing rules and so on), and a parser script provides the parser engine the necessary information to extract the relevant data. We defined a simple but efficient parsing language which allows to search strings, maintain a cursor over a string, extract substring, delete or rewrite them. The language also provides some constructs such as *if*, *until*, and *jump*. The language processes the search results in a fast and efficient way (see Fig. 4).

3. EXPERIMENTAL RESULTS

Experiments were conducted on a dual PIV 2.60Ghz, 1.5Gb of RAM memory and a 100Mbps internet connection. In all our experiments the resources usage was negligible. Due to space constraints, we report only a subset of our results.

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 – a total of 600 search results – took 12.4 seconds. This test was done using *wget*, a common http retriever tool. Helios can retrieve and parse in parallel the same results in 4.6 seconds (see Fig 2). This time was largely dominated by Ask/Teoma, which displays a maximum of 15 results per page, obligating Helios to request 7 subsequent pages to obtain the desired top 100 results. To overcome this limitation, Helios can be configured to exploit both parallelism among engines and within a single engine.

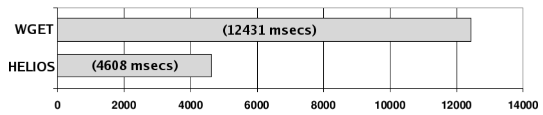

Figure 2: Helios vs wget. Query "madonna".

Parallel searches on single engine: Exploiting parallelism within a single engine can reduce the downloading time. Downloading from Ask/Teoma 7 search results pages required about 5.7 seconds. Exploiting parallel requests reduced this time to less than 1 second (see Fig. 3).

Parsing time: Helios parsed a total of 300 results from Altavista, Google and Yahoo! in less than 16 milliseconds (see Fig. 4).

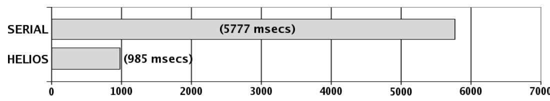

Figure 3: 7 pages from Ask/Teoma. Query "car".

Overall performances: Helios required less than 20 seconds to retrieve and parse 3000 results - 10 pages of 100 results from Altavista, Google and Yahoo!- (See Fig. 5).

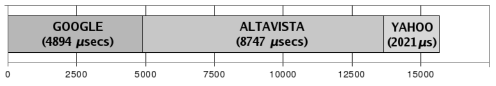

Figure 4: Parsing times per engine. Query "god".

Retrieving and parsing the top 200 results from Altavista, Google and Yahoo! – a total of 600 results – required only 2.21 secs, using 6 parallel connections. In comparison, Google required 2.39 secs to return its top 600 results using 6 parallel connections (see Fig. 6).

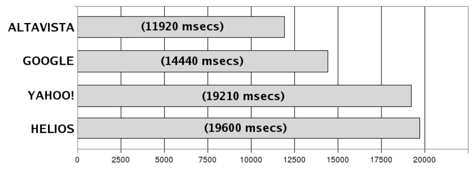

Figure 5: Retrieving and parsing times for 3000 URLs. Query "flowers".

Processing in parallel the first 100-results pages 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 and a 3% of average CPU usage. The query was "madonna".

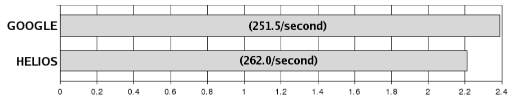

Figure 6: Retrieving time for Google and Helios. Time in secs. The query was "banana".

The experimental results show that Helios is an highly engineered open-source parallel meta-search engine. We can safely state that Helios can be used in industrial environments.

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] Chidlovskii. System and method for improving answer relevance in meta-search engines. U.S. Pat. 6829599, 2004.
[6] R.Fagin, R.Kumar, M.Mahdian, D.Sivakumar, and E.Vee. Comparing and aggregating rankings with ties. *PODS*, 2004.
[7] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. *www14*, 2005.
[8] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *www14*, 2005.
[9] S. Lawrence and C. L. Giles. Inquirus, the NECI meta search engine. In *WWW7*, 1998.
[10] W. Meng, C. Yu, and K. Liu. Building efficient and effective metasearch engines. In *ACM Computing Surveys*, 2002.
[11] M. E. Renda and U. Straccia. Web metasearch: Rank vs. score based rank aggregation methods. In *SAC*, 2003.
[12] F. Gibb S. Wu, F. Crestani. New methods of results merging for distributed information retrieval. In *Distributed Multimedia Information Retrieval*, 2003.
[13] R. Stevens. *UNIX Network Programming II*, Prentice Hall.