Search Engines for the World Wide Web: An Evaluation Methodology and their Optimization

Jagrati Jain
Student, Dronacharya College Of Engineering, Gurgaon

***Abstract*— For hundreds of years the mankind has organized information in order to make it more accessible to the others. The last media born to globally provide information is the Internet. With the Web, in particular, the name of the Internet has spread all over the World. Due to its impressive size and its high dynamicity, when we need to search for information on the Web, usually we begin by querying a Web Search Engine. A Web Search Engine maintains and catalogs the content of Web pages in order to make them easier to find and browse. Even though the various Search Engines are similar, each one of them differentiates from the other by the methods for scouring, storing, and retrieving information from the Web. Usually Search Engines search through Web pages for specified keywords. In response they return a list containing those documents containing the specified keywords. This list is sorted by relevance criteria which try to put at the very first positions the documents that best match the user's query. The usefulness of a search engine to most people, in fact, is based on the relevance of results it gives back. This paper tries to explain the functioning of web search engines, their internal structure and develop a search enginewhich ranks the document according to the searched keyword**

***Index Terms*— Search Engine,search engine for the web, search engine methodology, search engine optimization.**

I. INTRODUCTION

In 2008, Google reported that they had discovered over 1 trillion unique URLs on the Web. And as previous research has shown, it is unlikely that Google, or any other search engine, is even close to discovering all the available content on the Web. The Web is certainly a large place, and finding information can be a daunting task without a web search engine. In this article, we will examine how search engines work by examining the collection process, indexing of the content, and the factors that play in ranking the content.

Web Crawling

Web crawler download a web page, examine it for links to other pages, and continue downloading links it discovered until there were no more links left to be discovered.Figure 1.1 below shows how a web crawlerpulls from the Web and places downloaded web resources into a local repository. The next section will examine how this repository of web resources is then indexed and retrieved when you enter a query into a search engine.

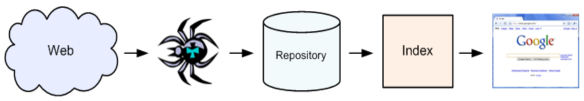

Figure 1.1: The Web is crawled and placed into a local repository where it is indexed and retrieved when using a search engine.

Indexing and Ranking

When a web crawler has downloaded a web page, the search engine will index its content. Often the **stop words**, words thatoccur very frequently like a, and, the, and to, are ignored. Other words might bestemmed. **Stemming** is a technique that removes suffixes from a word to improve the content of the index. For example, eating, eats, and eaten may all be stemmed to eat so that a search for eat will match all its variants.

An example index (usually called an **inverted index**) will look something like this where the number corresponds to a web page that contains the text:

cat > 2, 5
dog > 1, 5, 6
fish > 1, 2
bird > 4

So a query for *dog* would result in pages 1, 5, and 6. A query for *cat dog* would only result in page 5 since it is the only page that contains both search terms. Some search engines provide advanced search capabilities, so a search for *cat OR dog and NOT fish* would be entered which would result in pages 1, 5, and 6.

The search engine also maintains multiple weights for each term. The weight mightcorrespond to any number of factors that determines how relevant the term is to its host web page. **Term frequency** is one such weight which measures how often a term appears ina web page. Another weight that is given to a web page is based on the context in which the term appears in the page. If the term appears in a large, bold font or in the title of the page, it may be given more weight than to a term that appears in a regular font. A page might also be given more weight if links pointing to the page use the term in its anchor text. A final weight which most search engines will use is based on the web graph, the graph which is created when viewing web pages as nodes and links as directed edges. Good pages receive many citations, and bad pages receive few. So pages that have in-links from many other pages are probably more important and should rank higher than pages that few people link to. Brin and Page named their ranking algorithm **PageRank**, and it was instrumental in popularizing their new search engine called Google. All search engines today take into account the web graph when ranking results.

Figure 1.2 shows an example of a web graph where web pages are nodes and links from one page to another are directed edges. The size and color of the nodes indicate how much PageRank the web pages have. Note that pages with high PageRank (red nodes) generally have significantly more in-links than do pages with low PageRank (green nodes).

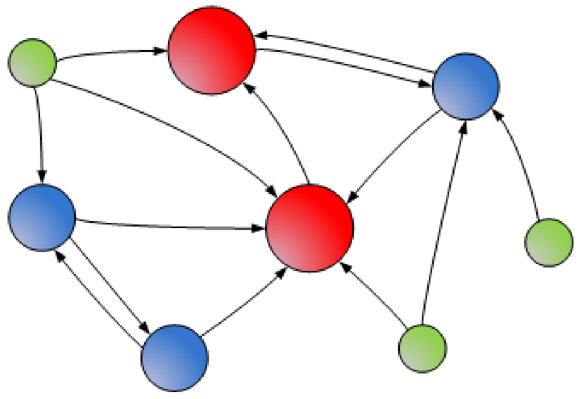

Figure 1.2: Example web graph. Pages with higher PageRank are represented with larger nodes.

Ranking Optimization

Search engines guard their weighting formulas as a trade secret since it differentiates their service from other search engines, and they do not want content-producers (the public who produces web pages) to "unfairly" manipulate their rankings.However, many companies rely heavily on search engines for recommendations and customers, and their ranking on a **search engine results page (SERP)** is very important. An industry based on **search engine optimization (SEO)** thrives on improving their customer's rankings by designing their pages to maximize the various weights discussed above and to increase the number and quality of incoming links.

Caching Search Engine Query Results

WSE caching, similarly to Web page caching, can occur at several places, e.g. on the client side, on a proxy, or on the server side. Caching on either the client or the proxy has the advantage of saving network bandwidth. Caching on the server side, on the other hand, has the advantage of improving the shareness of query results among different users. Moreover, caching at this level has the effect of saving I/O and computational resources used by the WSE to compute the page of relevant results to be returned to a user. In fact, consider that, in order to prepare a page of results, we have to intersect inverted lists that can be distributed, and to globally rank the results to decide which are the most relevant ones. On the other hand, cache results are cheaper to retrieve since it, usually, involves just a look-up operation on a search data structure.

One of the issues related to server-side cache is the limited resources usually available on the WSE server, in particular the RAM memory used to store the cache entries. However, the architecture of a scalable, large-scale WSE is very complex and includes several machines which take care of the various sub-tasks involved in the processing of user queries. Figure 1.3 shows the typical architecture of a modern large-scale WSE placed behind an http server. We can see a distributed architecture composed by a farm of identical machines running multiple WSE CORE modules, each of which is responsible for searching the index relative to one specific sub-collection of documents. This organization of the index is called *Local Inverted File* or *Document Partitioning*, in contrast to a *Global Inverted File* of *Term Partitioning* in which a complete index is horizontally split so that different index partitions refer to a subset of the set of distinct terms of the collection. In front of these searcher machines we have an additional machine hosting the MEDIATOR/BROKER. This module has the task of scheduling the queries to the various searchers, and of collecting the results returned back. The mediator then orders these results on the basis of their relevance, and produces a ranked vector of document identifiers (DocIDs), e.g. a vector composed by 10 DocIDs. These DocIDs are then used to get from the URLS/SNIPPETS SERVER the associated URLs and page snippets to include in the html page returned to the user through the http server.

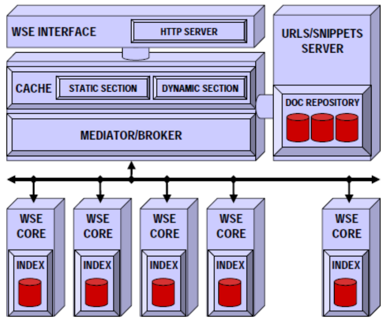

Figure 1.3: Architecture of a large-scale, distributed WSE hosting a query results cache.

Here we are interested in studying the design and implementation of such a server-side cache of query results. In particular, we will analyze the performance of a one-level cache in terms of hit-ratio and throughput. Starting from an accurate analysis of the content of three real query logs, we propose a novel replacement policy (called SDC- Static and Dynamic Cache) to adopt in the design of a fully associative cache of query results. According to SDC, the results of the most frequently accessed queries are maintained in a fixed size set of statically locked cache entries. This set is called *Static Set* an it is rebuilt at fixed time intervals using the statistical data coming from the WSE usage data. When a query arrives at SDC if it cannot be satisfied by the Static Set the it competes for the use of a *Dynamic Set* of cache entries.

We experimentally demonstrated the superiority of SDC over other caching policies proposed elsewhere, by evaluating both the hit-ratio and the throughput achieved on the three query logs by varying the size of the cache, the percentage of cache entries of the Static Set, and the replacement policy used for managing the Dynamic Set. Moreover, we showed that WSE query logs exhibit not only temporal locality, but also a limited spatial locality, due to requests for subsequent pages of results. Furthermore, our caching system exploits a sort of *Speculative Prefetching Strategy* that, differently from other prefetching proposals, tries to maintain a low overhead on the underlying WSE Core. In fact, while server-side caching surely reduces the load over the core query service of a WSE and improves its throughput, prefetching aims to increase the cache hit-ratio and thus the responsiveness (from the point of view of each single user) of the WSE, but may involve a large overhead on the same core query service. So an accurate study of tradeoffs of prefetching is necessary, and we addressed this in the experimental section of the chapter by analyzing pros and cons of different prefetching strategies.

In this work we will propose, in fact, SDC a novel caching policy which at the same cost of the *LRU* (or *SLRU*) policy (or even lower) will obtain hit-ratios that, in many cases, are better than those of *PDC*. In particular, we will see that, in principle, SDC can be combined with any other existing policies. In fact, with all the policies we considered in our experiments, SDC has always brought to performance enhancements. Furthermore, differently from other works which used only one log coming from a single WSE, we validated our results on three different query logs, coming from three different WSE, and referring to three different periods of time: a single day, a week, and a month.

<u>Analysis of the query logs</u>

In order to evaluate the behavior of different caching strategies we used query logs from the Tiscali, EXCITE, and Altavista search engines. In particular we used *Tiscali*, a trace

Query Log	Queries	Distinct Queries	Date
Excite	2,475,684	1,598,908	Sep 26th 1997
Tiscali	3,278,211	1,538,934	Apr 2002
AltaVista	6,175,648	2,657,410	A week of 2001

Table 1.1: Main characteristics of the query logs used.

of the queries submitted to the Tiscali WSE engine (www.janas.it) on April 2002, *Excite*, a publicly available trace of the queries submitted to the EXCITE WSE (www.excite.com) on September 26th 1997, and *Altavista* a query log containing queries submitted to Altavista on a week of 20024. Each record of a query log refers to a single query submitted to the WSE for requesting a *page* of results, where each page contains a fixed amount of URLs ordered according to a given rank. All query logs have been preliminarily cleaned by removing the useless fields. At the end of this preprocessing phase, each entry of a query log has the form *(keywords, page no)*, where *keywords* corresponds to the list of words searched for, and *page no* determines which page of results is requested. We further normalize the query log entries by removing those referring to requests of more than 10 results-per-page. In particular, since a WSE globally reorders the results according to a given rank, the top 10 results will be included in page 1, the next 10 results in page 2, and so on.

Table 1.1 reports the main characteristics of the query logs used. While about the 46% of the total number of queries appearing in the relatively recent Tiscali and Altavista logs are distinct, in the Excite log this percentage increases up to 64%. Therefore, only looking at the numbers of distinct queries appearing in the three logs, we could deduce that the locality found in the Excite log, i.e. the oldest one, might be less than in the Tiscali ones, since only the 36% (about 54% in the Tiscali and Altavista logs) of all its queries corresponds to re-submissions of previously submitted queries.

The SDC policy

In this section we describe SDC (Static and Dynamic Cache) a novel caching policy. Actually, this is a work extending a previously presented research. The idea that drove the entire design process is the following: *Is it possible to find a policy suitable for caching data which appear in accordance with a Zipf's law distribution, and having a time complexity equal to that of* LRU/SLRU*?*

SDC is a two-level policy which makes use of two different sets of cache entries. The first level contains the so called *Static Set*. It consists of a set of statically locked entries filled with the most frequent queries appeared in past users' sessions. The Static Set is periodically refreshed. The second level contains the *Dynamic Set*. Basically, it is a set of entries managed by a classical replacement policy (i.e. *LRU*, *SLRU*, etc.).

The behavior of SDC in the presence of a query q is, thus, very simple. First it looks for q in the Static Set, if q is present it returns the associated page of results back to the user. If q is not contained within the Static Set then it proceeds by looking for q in the Dynamic Set. If q is not present, then SDC asks the WSE for the page of results and replaces a page according to the replacement policy adopted.

Note that the idea of using a statically locked cache is present also in the work from Markatos where he studied a pure static caching policy for WSE results and compared it with a pure dynamic ones.

The rationale of adopting a static policy, where the entries to include in the cache are statically decided, relies on the observation that the most popular queries submitted to WSEs do not change very frequently. On the other hand, several queries are popular only within relatively short time intervals, or may become suddenly popular due to, for example, un-forecasted events (e.g. the 11th September 2001 attack). Basically, if the queries are distributed following a sort of *Zipf's law* behavior, then we may statically identify a set of queries to insert in the first level of the cache.

The advantages deriving from this novel caching strategy are two-fold. First, SDC present many interesting capabilities achieving the main benefits of both static and dynamic caching. In fact:

• the results of the most popular of the queries can always be retrieved from the Static Set even if some of these queries might be not requested for relatively long time intervals;

• the Dynamic Set of the cache can adequately cover sudden interests of users. Second, SDC may enhance performance. In fact, since accesses in the read-only section can be made concurrently without synchronization, this would eventually bring to good performance in a multi-threading environment.

Implementation Issues

First level - Static Set

The implementation of the first level of our caching system is very simple. It basically consists of a lookup data structure that allows to efficiently access a set of fstatic · N entries, where N is the total number of entries of the whole cache, and fstatic the factor of locked entries over the total. fstatic is a parameter of our cache implementation whose admissible values ranges between 0 (a fully dynamic cache) and 1 (a fully static cache). The static cache has to be initialized off-line, i.e., with the results of most frequent queries computed on the basis of a previously collected query log.

Each time a query is received; SDC first tries to retrieve the corresponding results from the Static Set. On a cache hit, the requested page of results is promptly returned. On a cache miss, we also look for the query results in the Dynamic Set.

Second level - Dynamic Set

The Dynamic Set relies on a replacement policy for choosing which pages of query results should be evicted from the cache as a consequence of a cache miss and the cache is full. Literature on caching proposes several replacement policies which, in order to maximize the hit-ratio, try to take the largest advantage from information about recency and frequency of references. SDC surely simplifies the choice of the replacement policy to adopt. The presence of a static read-only cache, which permanently stores the most frequently referred pages, makes in fact recency the most important parameter to consider.

As a consequence, some sophisticated (and often computationally expensive) policies specifically designed to exploit at the best both frequency and recency of references are probably not useful in our case. However, since we want to demonstrate the advantage of the SDC policy over the others, we implemented some of these sophisticated replacement policies. Currently, our caching system supports the following replacement policies: *LRU*, *LRU/2* which applies a *LRU* policy to the penultimate reference, *FBR*, *SLRU*, *2Q*, and *PDC* which consider both the recency and frequency of the accesses to cache blocks.

The choice of the replacement policy to be used is performed at start-up time, and clearly affects only the management of the (1−fstatic) ·N dynamic entries of our caching system.

Hereinafter we will use the following notation to indicate the different flavor of SDC. We will use SDC-rs to indicate SDC with replacement policy *r*, fstatic = s. For example: SDC-*LRU*0.4 means we are referring to SDC using *LRU* as the replacement policy, and fstatic = 0.4. Another example could be the following: SDC-[*LRU*/*SLRU*]0.4 which indicate SDC with a replacement policy chosen among *LRU*, and *SLRU*. Moreover, we will use the jolly character *, to indicate all the possible choice for the parameter replaced by *. So, SDC-*s will indicate SDC with any replacement policy and fstatic = s; while, SDC-p_ will indicate SDC with the replacement policy *p* and any value of fstatic.

The SI unit for magnetic field strength H is A/m. However, if you wish to use units of T, either refer to magnetic flux density B or magnetic field strength symbolized as μ0H. Use the center dot to separate compound units, e.g., ―A·m^2.‖

II. CONCLUSION

We are currently exploring several ways of improving our topic-sensitive PageRank approach. As discussed previously, discovering sources of search context is a ripe area of research. An- other area of investigation is the development of the best set of basis topics. For instance it may be worthwhile to use a finer-grained set of topics, perhaps using the second or third level of directory hierarchies, rather than simply the top level. However, a fine-grained set of topics leads to additional efficiency considerations, as the cost of thenaive approach to computing these topic-sensitive vectors is linear in the number of basis topics.

ACKNOWLEDGMENT

I would like to express my sincere gratitude to all those people who have given their heart willing support in making this completion a magnificent experience.

I am thankful to **Ms. Megha Goel**, H.O.D, Department of Technology, Dronacharya College of Engineering, Gurgaon for providing us good and healthy environment for the preparation of this dissertation.

I am also thankful to my dissertation guide **Mr. Vinod Kumar, Assistant Prof.**, Department of Information Technology for his timely comments and suggestions. He advised on the details of my dissertation and gave invaluable discussions. Without the guidance of my supervisor, this dissertation may not have well materialized.

I am really grateful to my parents for their support, appreciation and encouragement. A very special acknowledgement to authors of various research papers and books which helped me a lot.

REFERENCES

[1] The Google Search Engine: Commercial search engine founded by the originators of PageRank. http://www.google.com/.

[2] The Open Directory Project: Web directory for over 2.5 million URLs. http://www.dmoz.org/..

[3] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Stanford Digital Libraries Working Paper, 1998.

[4] Robert M. Gray and David L. Neuho_. Quantization. IEEE Transactions on Information Theory, 44(6), October 1998.

[5] Krishna Bharat and Monika R. Henzinger. Improved algorithms for topic distillation in a hyper- linked environment. In Proceedings of the ACM-SIGIR, 1998.

[6] Sergey Brin, Rajeev Motwani, Larry Page, and Terry Winograd. What can you do with a web in your pocket. In Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998.

[7] Sergey Brin and Larry Page. The anatomy of a large-scale hypertextual web search engine. In Proceedings of the Seventh International World Wide Web Conference, 1998.

[8] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In Proceedings of the Seventh International World Wide Web Conference, 1998.

[9] Jon Kleinberg. Authoritative sources in a hyperlinked environment. In Proceedings of the ACM- SIAM Symposium on Discrete Algorithms, 1998.

[10] Andrew McCallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In AAAI-98 Workshop on Learning for Text Categorization, 1998.

[11] Ian H. Witten, Alistair Mo_at, and Timothy C. Bell. Managing Gigabytes. Morgan Kaufmann Publishers, San Francisco, 1999.

[12] Krishna Bharat and George A. Mihaila. When experts agree: Using non-a_liated experts to rank popular topics. In Proceedings of the Tenth International World Wide Web Conference, 2001.

[13] Taher H. Haveliwala. Topic-sensitive PageRank. In Proceedings of the Eleventh International World Wide Web Conference, May 2002.

[14] Soumen Chakrabarti. Mining the Web: Discovering Knowledge from Hypertext Data. Morgan- Kaufmann Publishers, San Francisco, CA, 2002.

[15] Matthew Richardson and Pedro Domingos. The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank, volume 14. MIT Press, Cambridge, MA, 2002.

[16] Soumen Chakrabarti, Mukul M. Joshi, Kunal Punera, and David M. Pennock. The structure of broad topics on the web. In Proceedings of the Eleventh International World Wide Web Conference, 2002.

[17] Michelangelo Diligenti, Marco Gori, and Marco Maggini. Web page scoring systems for horizontal and vertical search. In Proceedings of the Eleventh International World Wide Web Conference, May 2002.

[18] Ronald Fagin, Ravi Kumar, and D. Sivakumar. Comparing top k lists. In Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 2003.

[19] Glen Jeh and Jennifer Widom. Scaling personalized web search. In Proceedings of the Twelfth International World Wide Web Conference, May 2003.

[20] Sepandar D. Kamvar, Taher H. Haveliwala, Christopher D. Manning, and Gene H. Golub. Ex- trapolation methods for accelerating PageRank computations. In Proceedings of the Twelfth International World Wide Web Conference, May 2003.

[21] Mike Thelwall. Quantitative Comparison of Search Engine Results, School of Computing and Information Technology, 2008.

[22] Web Search Engines – A Comparative Study by Inderjeet Singh Oberoi and Mridul