ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 7, July 2015

Web Optimization Using Web Mining Technique

Mahendra Pratap Singh Dohare[1], Dr. R S. Jadon[2]

Ph. D. Scholar, Dept of Computer Application, Madhav Institute of Technology & Science, Gwalior[1]

Prof& HOD, Dept of Computer Application, Madhav Institute of Technology & Science, Gwalior[2]

Abstract: A Web Optimization maintains and catalogs the content of Web pages in order to make them easier to find. The importance of WPO is only rising, as well as it grows, the need for tools that can assist developers in making the right also decisions grows. Hence that is the goal of this thesis: to build a tool that can be used for the continuous profiling of a web site's performance. 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 a relevance criterion which tries to put at the very first positions the documents that best match the inquiry of user. In meticulous, since the size of the Web is quickly rising, the central issues observe elevated presentation algorithms for information management. Furthermore, nowadays Web Optimizations receive more searches per day over a collection of several billion web pages indexed. These particular, can easily explain why in such environments the efficiency, as the effectiveness, of Search and Index algorithms have issues became. Intended for this manner in this paper we are going toward proposing novel techniques aimed at enhancing the performance of a Web Optimization from different angles.

Keywords: Web Mining, Web Optimization, k-means Algorithm.

I. INTRODUCTION

Web search services have proliferated in the last years. Users have to deal with different formats for inputting queries, different results presentation formats, and, especially, differences in the quality of retrieved information. Also performance (i.e. search and retrieval time plus communication delays) is a problem that has to be faced while developing such a type of application which may receive thousands of requests at the same time. Most search engine developments is done within competitive companies which do not publish technical details. Parallel and Distributed processing is an enabling technology for efficiently searching and retrieving information on the Web. Despite this fact, enhancements to sequential IR methods are very important. Various access methods have been developed to support efficient search and retrieval over text document collections. Inverted files have traditionally been the index structure of choice for the Web. Commercial search engines use custom network architectures and high–performance hardware to achieve sub–second query response times using such inverted indexes. When the collection is small and indexing is a rare activity, optimizing index–building is not as critical as optimizing run–time query processing and retrieval. However, with a Web scale index, and it build time also became a critical factor for two reasons: Scale and growth rate, and Rate of change. Popular search engines receive millions of queries daily, a load never experienced before by any IR system. Also search engines have to deal with a growing number of Web pages to discover, to index, and to retrieve information. To compound the problem, search engine users want to experience small response times as well as precise and relevant results for their queries. In this system, the development of techniques to improve the performance and the scalability of the results becomes a fundamental topic of research in IR. One effective alternative for improving performance and scalability of information systems is caching. The effectiveness of caching strategies depends on some key aspects, such as the presence of reference locality in the access stream, and the frequency at which the database is being cached and updated. Users usually look only at the very first pages returned by a Web Optimization, it is very important to effectively rank the results returned for the submitted queries. The two main techniques used in ranking algorithms for Web pages are: Statistical (i.e. based on words occurrence in the pages), and Link Based (i.e. based on importance inferred from information on the structure of the Web graph). In this paper we focus our attention on link based techniques [1] and [3-5].

II. LITERATURE SURVEY

Various access methods have been developed to support efficient search and retrieval over text document collections. Inverted files have traditionally been the index structure of choice for the Web. Commercial search engines use custom network architectures and high–performance hardware to achieve sub–second query response times using such inverted indexes. When the collection is small and indexing is a rare activity, optimizing index–building is not as critical as optimizing run–time query processing and retrieval. However, with a Web–scale index, index build time also became a critical factor for two reasons: Scale and growth rate, and Rate of change. An inverted list implementation that supports jumping forward in the compressed list using skip pointers. This is useful for document based access into the list during conjunctive style processing. The purpose of these skip pointers is to provide synchronization points for decompression, allowing just the desired portions of the inverted list to be decompressed. Data structure to perform efficient on-line update of the index with low performance loss. The structure they use is organized as block of disk storage allocated for index blocks. Together, the index blocks make up the index, that combines the functions of the dictionary and postings file, thus the index contains both index terms and postings into a single file. Updates are buffered in main memory until they can be applied to disk. A background process continuously cycles through index storage applying updates and re-writing the index. Update throughput is thus a function of the size of the main memory buffer and the period of an update cycle [3] and [5].

Popular search engines receive millions of queries daily, a load never experienced before by any IR system. Additionally, search engines have to deal with a growing number of Web pages to discover, to index, and to retrieve. To compound the problem, search engine users want to experience small response times as well as precise and relevant results for their queries. In this scenario, the development of techniques to improve the performance and the scalability of the results becomes a fundamental topic of research in IR. One effective alternative for improving performance and scalability of information systems is caching. The effectiveness of caching strategies depends on some key aspects, such as the presence of reference locality in the access stream, and the frequency at which the database is being cached and updated. Caching is a very effective technique to make scalable a service that distributes data to a multitude of clients. As suggested by many researchers, caching can also be used to improve the efficiency of a Web Optimization. This is motivated by the high locality present in the stream of queries processed by a Web Optimization, and by the relatively infrequent updates of Web Optimization indexes that allow us to think of them as mostly read-only data.

III. TECHNIQUES

3.1 Parallel Crawling

As the size of the Web grows, it becomes more difficult to retrieve the whole or a significant portion of the Web using a single process. Therefore many Web optimizations run multiple crawler processes in parallel. We refer to this type of crawler as a parallel crawler. Only few works discuss the architecture of parallel crawlers. Heydon and Najork describe the software architecture of Atrax the distributed version of Mercator: a scalable and extensible web crawler. Crawling is performed by multiple worker threads. Each thread repeatedly performs the steps needed to download and process a document. All Mercator threads are run in a single process. However, Mercator can be configured as a multi–process distributed systems. In this configuration, one process is designated the queen, and the others are drones. Both the queen and the drones run worker threads, but only the queen runs a background thread responsible for logging statistics, terminating the crawl, and initiating checkpoints. Moreover, in its distributed configuration the state of a Mercator crawl is fully partitioned across the queen and drone processes; there is no replication of data. In a distributed crawl, when a link has been extracted it is checked to see if it is assigned or not to this process. If not it is routed to the appropriate peer process. Since about 80% of links are relative, the vast majority of discovered URLs remain loyal to the crawling process that discovered them. Mercator was written in Java, which gives flexibility through pluggable components but also posed a number of performance problems that have been addressed by the authors.

Various challenges are in the development of an effective Crawler. In particular he had addressed the parallelization of the Crawling phase. The goal is to propose some guidelines for crawler designers, helping them selecting operational parameters like: number of crawling processes, or inter-process coordination and communication schemes. The author considers a general architecture of a parallel crawler as composed by several crawling processes named C-procs. Each C-proc performs the basic tasks that a single-process crawler conducts. The C-procs may be running either on a LAN (Intra-site Parallel Crawler) or on the Internet [12-15].

The software architecture consists of a number of agents, each one delegated to deal with a specified portion of the web domain under investigation. The main components of the crawler are: the Store that deals with the storage of the crawled pages and the checking of the duplicates; the Frontier that retrieves new pages on the basis of the actually fetched pages; the Controller that serves as crash-failure detector. The uses of mobile agents are to improve the performance of Web Optimizations. The performance gains translate to improved web coverage and freshness of search results. The proposed approach was consisting of uploading a software agent to participating servers and using this agent to collect pages and sending them to the search engine site. The authors also give explanations of the security issues related to this approach and show that, due to its simplicity; their proposal does not introduce new security concerns. Moreover, security can be enforced by simple conventional techniques which are computationally inexpensive [8] and [10].

3.2 Parallel Indexing

Despite their simple structure, the task of building inverted files for very large text collections such as the Web is very expensive. Therefore, faster indexing algorithms are always desirable and the use of parallel or distributed hardware for generating the index is an obvious solution.

An important feature of the IF index organization is that indices generated following this can be easily partitioned. In particular depending on the moment the partitioning phase is done one can devise two different partitioning strategies. The first approach requires to horizontally partition the whole inverted index with respect to the lexicon, so that each query server stores the inverted lists associated with only a subset of the index terms. This method is also known as term partitioning or global inverted files. The other approach, known as document partitioning or local inverted files, requires that each query server becomes responsible for a disjoint subset of the whole document collection (vertical partitioning of the inverted index). Following this last approach the construction of an IF index become a two-staged process. In the first stage each index partition is built locally and independently from a partition of the whole collection. The second phase is instead very simple, and is needed only to collect global statistics computed over the whole IF index [10] and [11] and [12].

3.3 Efficient Query Brokering
Common software architecture for parallel IRSs follows the Master/Worker model. In this model the Workers are the actual search module which receive queries from and return results to the Master that, in this schema, is known as the Query Broker (QB). Since realistic Web Optimization usually manage distinct indexes, the only way to ensure timely and economic retrieval is designing the QB module so that it forward a given query only to the workers managing documents related to the query topic. The Collection Selection techniques play a fundamental role in the reduction of the search space. Particular attention should be paid in using this technique since it could results in a loss of relevant documents thus obtaining dramatic effectiveness performance degradations [13] and [15].

3.4 Indexing in Web Optimizations
Several sequential algorithms have been proposed, which try to well balance the use of core and out-of-core memory in order to deal with the large amount of input/output data involved. The Inverted File (IF) index is the data structure typically adopted for indexing the Web. This is mainly due to two different reasons. The first is that an IF index allows the resolution of queries on huge collections of Web pages to be efficiently managed, and works very well for common Web queries, consisting of the conjunction of a few terms. An IF index on a collection of Web pages consists of several interlinked components. The principal ones are: the lexicon, i.e. the list of all the index terms appearing in the collection, and the corresponding set of inverted lists, where each list is associated with a distinct term of the lexicon. Each inverted list contains, in turn, a set of postings. Each posting collects information about the occurrences of the corresponding term in the collection's documents. For the sake of simplicity, in the following discussion we will consider that each posting only includes the identifier of the document where the term appears, even if postings actually store other information used for document ranking purposes that means in our implementation each posting also includes the positions and the frequency of the term within the document, and context information like the appearance of the term within specific html tags. Another important feature of the IF indexes is that they can be easily partitioned. In fact, let us consider a typical parallel query analyzer module: the index can be distributed across the different nodes of the underlying architecture in order to enhance the overall system's throughput. The two different partitioning strategies can be devised. The first approach requires to horizontally partition the whole inverted index with respect to the lexicon, so that each query server stores the inverted lists associated with only a subset of the index terms. This method is also known as term partitioning or global inverted files. The other approach, known as document partitioning or local inverted files, requires that each query server becomes responsible for a disjoint subset of the whole document collection (vertical partitioning of the inverted index). Following this last approach the construction of an IF index become a two-staged process. In the first stage each index partition is built locally and independently from a partition of the whole collection. The second phase is instead very simple, and is needed only to collect global statistics computed over the whole IF index [14] and [16].

III. PROPOSED TECHNIQUES

In Web Optimization we can identify three principal modules: the Spider, the Indexer, and the Query Analyzer. It can be exploit parallelism in all the three modules. For the Spider we can use a set of parallel agents which visit the Web and gather all the documents of interest. Furthermore, parallelism can be exploited to enhance the performance of the Indexer, which is responsible for building an index data structure from the collection of gathered documents to support efficient search and retrieval over them. Finally, parallelism and distribution is crucial to improve the throughput of the Query Analyzer, which is responsible for accepting user queries, searching the index for documents matching the query, and returning the most relevant references to these documents in an understandable form.

4.1 Indexing in Web Optimizations-

Several sequential algorithms have been proposed, which try to well balance the use of core and out-of-core memory in order to deal with the large amount of input/output data involved. The Inverted File (IF) index is the data structure typically adopted for indexing the Web. This is mainly due to two different reasons. The first is that an IF index allows the resolution of queries on huge collections of Web pages to be efficiently managed, and works very well for common Web queries, consisting of the conjunction of a few terms. Second, an IF index can be easily compressed to reduce the space occupancy
in order to better exploit the memory hierarchy. An IF index on a collection of Web pages consists of several interlinked components.
The principal ones are: the lexicon, i.e. the list of all the index terms appearing in the collection, and the corresponding set of inverted lists, where each list is associated with a distinct term of the lexicon. Each inverted list of contains, and turn a set of postings. Each posting collects information about the occurrences of the corresponding term in the collection's documents. For the sake of simplicity, in the following discussion we will consider that each posting only includes the identifier of the document where the term appears, even if postings actually store other information used for document ranking

purposes means in our implementation each posting also includes the positions and the frequency of the term within the document, and context information like the appearance of the term within specific html tags.
Another feature of the IF indexes is that they can be easily partitioned. In fact, let us consider a typical parallel Query Analyzer module: the index can be distributed across the different nodes of the underlying architecture in order to enhance the overall system's throughput the number of queries answered per each second. For this purpose, two different partitioning strategies can be devised. The first approach requires to horizontally partition the whole inverted index with respect to the lexicon, so that each query server stores the inverted lists associated with only a subset of the index terms. This method is also known as term partitioning or global inverted files. The other approach, known as document partitioning or local inverted files, requires that each query server becomes responsible for a disjoint subset of the whole document collection (vertical partitioning of the inverted index). Following this last approach the construction of an IF Index becomes a two-staged process. In the first stage each index partition is built locally and independently from a partition of the whole collection. The second phase is instead very simple, and is needed only to collect global statistics computed over the whole IF index.

4.2 We proposed two different assignment schemes:

• Top-down assignment: we start from the collection as a whole, and we recursively partition it by assigning, at each level, similar documents to the same partition. At the end of this partitioning phase a merging phase is performed until a single and ordered group of documents is obtained. The assignment function π is then deduced by the ordering of this last single group. This is the approach also followed by B&B. Within this scheme we propose two different algorithms which will be discussed in the following: TRANSACTIONAL B&B and Bisecting;
• Bottom-up assignment: we start from a flat set of documents and extract from this set disjoint sequences containing similar documents. Inside each sequence the documents are ordered, while we do not make any assumption on the precedence relation among documents belonging to different sequences. The assignment function π in this case is deduced by first considering an arbitrary ordering of the produced sequences and then the internal ordering of the sequences themselves. In our case to order the produced sequences we simply consider the same order in which the sequences are produced by the algorithms themselves. Within this approach we propose two different algorithms: single-pass k-means and k-scan.

4.3 Top-down assignment

➔Center selection: according to some heuristic H, we select two (groups of) documents from D which will be used as partition representatives during the next step;
➔ Redistribution: according to their similarity to the centers, we assign each unselected document to one of the two partitions D' and D''. Actually, we adopt a simple heuristic which consists in assigning exactly | D|/2 documents to each partition in order to equally split the computational workload among the two partitions;
➔Recursion: we recursively call the algorithm on the two resulting partitions until each partition becomes a singleton;
➔Margin: the two partitions built at each recursive call are merged (operator X-OR) bottom-up thus establishing an ordering (<=) between them. The precedence relation <= is obtained by comparing the borders of the partitions to merge (D' and 'D'') and, according to the distance measure adopted, we put D' before D'' if the similarity between the last document(s) of D' and the first document(s) of D'' is greater than the similarity computed by swapping the two partitions. It is also possible to devise a general cost scheme for such top-down algorithms.

The TRANSACTIONAL B&B algorithm is basically a porting under our model of the algorithm. It starts by computing a sampled similarity graph: it chooses a document out of $| D|^p$ (p is the document sampling factor 0 <p < 1) only considering terms appearing in less than documents. After this reduced similarity graph has been built, it applies graph partitioning algorithm, which splits the graph in two equally sized partitions. The algorithm than proceeds with the redistribution, recursion, and merging steps of the generic top-down algorithm. However, since in our model we do not have an IF index previously built over the document collection, we cannot know which terms appear in less than documents, and thus we did not introduce sampling over the maximum term frequency as in the original implementation.

4.4 Bisecting

The second algorithm we propose is called Bisecting. In this algorithm we adopt a center selection step which simply consists of uniformly choosing two random documents as centers. The cost of the centers selection step is thus reduced considerably. The algorithm is based on the simple observation that, since in TRANSACTIONAL B&B the cost CH may be high, the only way to reduce it is to choose a low sampling parameter p, thus selecting at each iteration a very small number of documents as centers of the partitions. Thus we thought to just get rid of the first three phases, i.e. sampling, graph building.

4.5 Bottom-up assignment

These algorithms consider each document of the collection separately, and proceed by progressively grouping together similar documents. Our bottom-up algorithms thus produce a set of non-overlapping sequences of documents. The two different assignment algorithms presented here are both inspired by the popular k-means clustering algorithm:

• A single-pass k-means algorithm;
• K-scan which is based on a centroid search algorithm which adapts itself to the characteristics of the processed collection.

4.6 K-means technique

K-means is a popular iterative clustering technique which defines a Centroid Voronoi Tessellation of the input space. The k-means algorithm works as follows. It initially chooses k documents as cluster representatives, and assigns the remaining |D'| - k documents to one of these clusters according to a given similarity metric. New centroids for the k clusters are then recomputed, and all the documents are reassigned according to their similarity with the new k centroids. The algorithm iterates until the position of the k centroids become stable. The main strength of this algorithm is the O (|D'|) space occupancy. On the other hand, computing the new centroids is expensive for large values of |D'|, and the number of iterations required to converge may be high. The single-pass k-means consists of just the first pass of this algorithm where the k centers are chosen using the technique "Buckshot". We will not describe here the Buckshot technique, the only thing to keep into account is that the complexity of this step do not influence the theoretical linear performance of k-means which remains O (k| D'|). Since the k-means algorithm does not produce ordered sequences but just clusters, the internal order of each cluster is given by the insertion order of documents into each cluster.

The other bottom-up algorithm developed is k-scan. It resembles to the k-means one. It is, indeed, a simplified version requiring only k steps. At each step i, the algorithm selects a document among those not yet assigned and uses it as centroid for the i-th cluster. Then, it chooses among the unassigned documents the |D'|/k - 1 ones most similar to the current centroid and assign them to the i-th cluster. The time and space complexity is the same as the single-pass k-means one and produces sets of ordered sequences of documents. Such ordering is exploited to assign consecutive DocIDs to consecutive documents belonging to the same sequence. It takes as input parameters the set D' and the number k of sequences to create. It outputs the ordered list of all the members of the k clusters. This list univocally defines π, an assignment of D' minimizing the average value of the d-gaps.

V. RESULTS ANALYSIS

We presented an analysis of several efficient algorithms for computing approximations of the optimal Doc ID assignment for a collection of textual documents. We have proved that our algorithms are a viable way to enhance the compressibility (up to 26%) of IF indexes. The algorithms proposed operate following two opposite strategies: a top-down approach and a bottom-up approach. The first group includes the algorithms that recursively split the collection in a way that minimizes the distance of lexicographically closed documents. The second group contains algorithms which compute an effective reordering employing linear space and time complexities. Although our algorithms obtain gains in compression ratios which are slightly worse than those obtained by the previous algorithm, their performance in terms of space and time are instead remarkably higher. We also wanted a classification of user. Which identify uses their own web access manner. We will try to improve accessibility time and provide better more relevant result according user requirement.
Just for the sake of completeness we must say that the actual B&B algorithm complexity is higher and this is mainly due to two reasons. First B&B needs an IF to be built in order to start its computations. Then it needs to load the entire IF into main memory before starting the actual reordering phase.

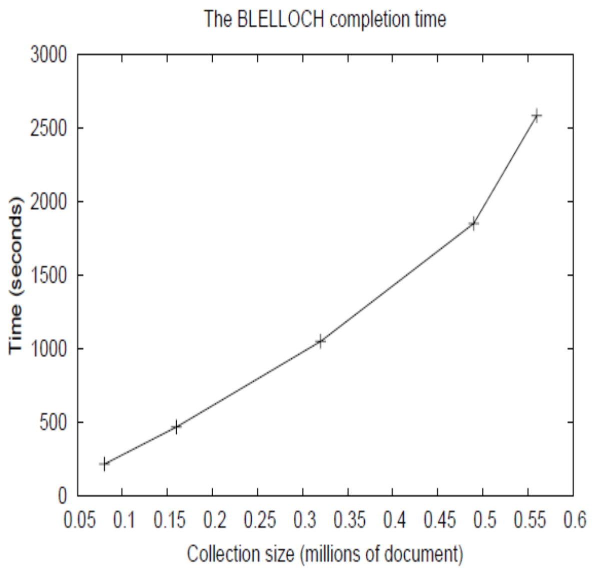

Figure: The scalability of the B&B algorithm with varying the size of the collection reordered.

To conclude, the B&B algorithm is very good for computing an optimal reordering of the document identifiers in order to reduce the space occupancy of a compressed IF but, to do so, it must first build a sort of Inverted List Index before starting. This step should be avoided in real Web Optimization Systems. The observations made above may bring us to consider the problem from another point of view. In fact we would not start from an already built IF. Instead, we would like to assign identifiers to documents as they arrive to the indexers.

V I. CONCLUSION AND FUTURE WORKS

The design and implementation, as well the analysis, of efficient, and effective Web Optimizations, are becoming more and more important as the size of the Web has continually kept growing. Furthermore, the development of systems for Web Information Retrieval represents a very challenging task whose complexity imposes the knowledge of several concepts coming from many different areas: databases, parallel computing, artificial intelligence, statistics, etc. In this paper three important issues related to Web optimization technology have been investigated. As future work we plan to test the performance of our algorithms on some recently proposed encoding methods. In particular we would like to evaluate the method for which we should be able to obtain good at most relevant results. Furthermore, we will want to investigate possible adaptations of the algorithms proposed to collections which change dynamically in the time.

REFERENCES

[1] Hiroyuki Kawano, "Overview of Mondou Web search engine using text mining and information visualizing technologies", IEEE 2001.
[2] Animesh Tripathy, Prashanta K Patra, "A Web Mining Architectural Model of Distributed Crawler for Internet Searches Using PageRank Algorithm" IEEE 2008.
[3] Sadegh Kharazmi, Ali Farahmand Nejad, Hassan Abolhassani." Freshness of Web Optimizations: Improving Performance of Web Optimizations Using Data Mining Techniques" IEEE 2009.
[4] Rowena Chau, Chung-Hsing Yeh, "Intelligent Techniques for Global E-Business Web Portal Development", Published IEEE Conference.
[5] Dan Blandford and Guy Blelloch. Index compression through document reordering. In IEEE, editor, Proceedings of the DATA COMPRESSION CONFERENCE (DCC'02). IEEE, 2002.
[6] V. N. Anh and A. Moffat. Index compression using fixed binary codewords. In K.- D. chewe and H.Williams, editors, Proc. 15th Australasian Database Conference, Dunedin, New Zealand, January 2004.
[7] M.W. Berry and M. Browne. Understanding Search Engines: Mathematical Modeling and Text Retrieval. SIAM Book Series: Software, Environments, and Tools, June 1999.
[8]S. Brin and L. Page. The Anatomy of a Large–Scale Hypertextual Web Search Engine. In Proceedings of the WWW7 conference / Computer Networks, volume 1–7, pages 107–117, April 1998.
[9] Soumen Chakrabarti. Mining the Web - Discovering Knowledge from Hypertext Data. Morgan Kaufmann Publishers, San Francisco, 2003.
[10] Junghoo Cho. Crawling the Web: Discovery and Maintenance of Large-Scale Web Data. PhD paper, Stanford University, October 2001.
[11] Junghoo Cho and Hector Garcia-Molina. The Evolution of the Web and Implications for an Incremental Crawler. In The VLDB Journal, pages 200–209, 2000.
[12] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide–Area Cooperative Storage with CFS. In Proceedings of the 18th ACM Symposium on Operating Systems Principles, October 2001.
[13] P. Elias. Universal codeword sets and representation of the integers. IEEE Transactions on Information Theory, 21(2):194–203, February 1975.
[14] T. Feder, R. Motwani, R. Panigrahy, S. Seiden, R. van Stee, and A. Zhu. Web caching with request reordering. In Proceedings of the 13th Annual Symposium on Discrete Algorithms, pages 104–105, 2002.
[15] Vijay Gupta and Roy Campbell. Internet Search Engine Freshness by Web Server Help. In Proceedings of the 2001 Symposium on Applications and the Internet (SAINT 2001). IEEE, 2001.
[16] B.S. Jeong and E. Omiecinski. Inverted File Partitioning Schemes in Multiple Disk Systems. IEEE Transactions on Parallel and Distributed Systems, (2), February 1995.