# A Graph-Based Indexing Technique to Enhance the Performance of Boolean AND Queries in Big Data Systems

Abdulla Kalandar Mohideen*, Shikharesh Majumdar*, Marc St-Hilaire* and A. El-Haraki†
*Dept. of Systems and Computer Engineering, Carleton University, Ottawa, Canada
†Telus, Ottawa, Canada
Email: {abdullakalandarmohid@cmail., majumdar@sce., marc_st_hilaire@}carleton.ca, ali.el-haraki@telus.com

***Abstract*—This paper introduces a new graph-based indexing (GBI) technique for big data systems. It uses a directed graph structure that effectively captures the simultaneous occurrence of multiple keywords in the same document. The objective is to use the relationship between the search keywords captured in the graph structure to effectively retrieve all results of Boolean AND queries at once. The performance of the proposed technique is compared with the conventional inverted index-based technique. This paper highlights that, irrespective of the intersection algorithm used to evaluate Boolean AND queries, GBI always returns Boolean AND search results faster than the inverted index. This is due to the fact that GBI always performs a smaller number of intersection operations and avoids intersection if search keywords do not have a common document. A preliminary performance analysis is performed through prototyping and measurement on a system subjected to a synthetic workload. The analysis shows that GBI improves search latency when executing Boolean AND queries by an average of 69% to 99.9% in comparison to the inverted index.**

***Index Terms*—text indexing, keyword search, Boolean queries, graph-based index**

## I. Introduction

It has been estimated that around 3.5 billion searches [1] have been made daily using the Google search engine. One of the key techniques that have been adopted to return results faster is to index documents. Search engines like Google[1] and Lucene[2] use the inverted index for enabling faster keyword searching. In [2], the authors have explained the basic concepts and terminology for an inverted index. The inverted index is a mapping of unique meaningful words extracted from the documents set to be indexed to their corresponding postings list. A document can be defined as a source of textual data from which meaningful unique keywords can be extracted. Examples of such documents include tweets and web pages. Each document is tagged with a unique document id. A postings list for a keyword is defined as a collection of document ids.

The authors in [3] illustrate different types of search queries. Out of all types of queries, Boolean queries are among the most commonly evaluated queries. Boolean queries are constructed using one of the Boolean operators (AND, OR, NOT) between search keywords and the most commonly used operator is the Boolean AND. The evaluation of Boolean AND queries are computationally expensive in the inverted index, especially when indexing millions of documents. In web search engines, where there is no need to obtain all the resultant document ids at once, techniques like page ranking have been used [4]. The web pages to be indexed have been given some ranking based on factors like popularity or number of hits and only the top $K$ results are retrieved where $K$ represents the number of resultant document ids. Similar to web page ranking, the authors of [5] demonstrated an optimized scoring mechanism to retrieve top-scored documents for the Boolean search queries.

The motivation for this research is to effectively evaluate Boolean AND queries for applications requiring all the results at once (not only the top $K$). In this case, adopting ranking concepts does not work since all matching documents ids need to be retrieved. Optimizing Boolean AND queries performance for retrieving all results at once is crucial for big data systems when they are deployed in the cloud. It helps in reducing the overall search response time of cloud systems. One solution to get all the matching document ids is to perform the intersection between the given search keywords postings lists. The authors of [6] studied various intersection algorithms based on how the data is stored. The study concluded that when data is stored as it is without any compression techniques, the traditional Small-Vs-Small (SvS) algorithm is the best way to do an intersection.

This paper proposes a new indexing scheme referred to as graph-based index (GBI) for big textual data systems to evaluate Boolean queries. GBI uses a directed graph data structure, represented using an adjacency matrix and is implemented using a hash table. The graph data structure can be accommodated in any real-time scenario where there is a need for establishing a link between two entities that have some common traits. Textual document indexing can be viewed as a relation between a keyword and the document it appears in. GBI treats different keywords that appear in a document as entities that are to be related by a common trait called document id. This results in establishing a direct relationship between a keyword and other keywords that appear in a document. Capturing this direct relationship between different keywords forms an innovative element for reducing the

[1]https://gofishdigital.com/googles-indexer-caffeine/
[2]https://lucene.apache.org/

978-1-7281-6095-5/20/$31.00 ©2020 IEEE
DOI 10.1109/CCGrid49817.2020.00-24

number of postings lists needed for an intersection operation in evaluating Boolean AND queries that requires all results at once. To the best of our knowledge, this is the first time that a graph structure exhibiting keyword-keyword-document ids relationship has been adopted for text indexing. The main contributions of this short paper include:

1) A novel graph-based index structure to effectively evaluate Boolean AND queries.
2) A preliminary proof of concept prototype for demonstrating the effectiveness of the proposed graph-based indexing technique.

The rest of this paper is organized as follows. Section 2 introduces the GBI structure and discusses the Boolean AND search using a GBI. Section 3 presents a performance evaluation of the GBI and compares its performance to the inverted index technique. Finally, Section 4 provides our conclusions and directions for future work.

## II. Graph-Based Index Structure

Textual data in a document subjected to indexing undergoes a keyword extraction process where the important, unique words are extracted from the text. The process involves one or all of the following three steps: lexing, stemming and stop words removal [3]. Lexing is the process of converting a list of characters to a list of tokens. Each such token is a single alphanumeric word. Each token is then transferred into its morphological root, a process called stemming. Stop word removal is the process of removing common words that almost occurs in any textual data like articles and prepositions to name a few. This keyword extraction is typically applied irrespective of the indexing method or tools used. In this short paper, it is assumed that the keywords for indexing are already obtained and therefore we focus on the strategy used to index these keywords in the GBI. It is also assumed that each document to be indexed is given a unique id.

Consider, for example, the document that contains following textual phrase to be indexed,

Document 1: "That were you, Anthony, the son of Caesar, You should be satisfied."

After applying the keyword extraction process on the document, the keywords "*Anthony*", "*Son*" and "*Caesar*" are obtained. First, the keywords are sorted in ascending order. Thus, the sorted keywords set becomes *Anthony*, *Caesar*, *Son*. Each keyword in the set represents a node in the GBI. Then, an edge containing the document id as weight is added from each keyword to the remaining set of keywords that appear after it in the sorted set. This process is repeated until the last keyword in the sorted set is reached. If the id of document 1 is 1 then the resulting GBI is presented in Figure 1. As can be seen, the graph stores relationships between keywords. For example, *Anthony* and *Caesar* are related by the document number 1. Thus, a quick search for "*Anthony* and *Caesar*" will evaluate the result to 1 without performing any intersection operation thereby saving a lot of query evaluation time. Algorithm 1 provides a high-level description of the indexing process in GBI.

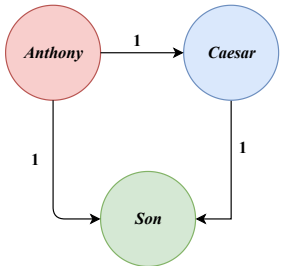


Fig. 1. GBI after indexing three keywords.

**Algorithm 1** Indexing in Graph-Based Index

**Input:** Set of keywords ($S$) and identifier ($Id$) of document
**Output:** New or updated hash table representing GBI ($G$)
**Condition:** At least two keywords exists in set $S$

```
Sort S in ascending order
for each entry key1 in S do
    if key1 is last entry in S then
        break
    end if
    for each entry key2 occurring after key1 in S do
        Create or update an entry in G using the hash table
        key [key1][key2] with Id as value
    end for
end for
```

### A. Boolean AND Search in Graph-Based Index

The Boolean AND search queries are a type of queries where all the search terms specified must be found in a document to qualify it to be a part of the resultant set. In other words, let *n1, n2, ..., nk* be the search terms, *R* be the resultant set containing ids of the matching documents and $D1$ be the document that is currently being examined. Then, *D1* is qualified to be part of *R* if and only if all the search terms *n1, n2, ..., nk* are present in document *D1*. It is important to note that the position in which the search keywords appear within the document is irrelevant. The only condition is that all the search terms should appear in the document to be part of the resultant set of the queries.

Consider the following example where the keyword *Anthony* appears in documents *D1, D2* and *D5* and the keyword *Caesar* appears in documents *D1, D2, D3* and *D4*. Then, a Boolean AND query looking for conjunction yields the result *D1* and *D2*.

The working of the Boolean AND search on GBI is described as follows. In general, let $S$ be the set containing the search keywords. Initially, the set $S$ is sorted in ascending order. Each keyword except the last keyword in set $S$ is paired with the next occurring keywords to find whether there exists an edge between them. The document ids are collected only if keyword pairs are mutually exclusive or it is the last keyword pair in $S$. This is to reduce the total number of keyword pairs needed to compute the intersection between them. To illustrate this, consider four keywords *A, B, C* and *D*. Since

the keywords are indexed in pairs (i.e. *AB, AC, AD, BC, BD*, and *CD*) only non-overlapping keyword pairs (in this case AB and CD) are needed to compute the intersection. Therefore, only the document ids of these two pairs are collected. Now, let's consider five search keywords say *A, B, C, D* and *E*. The number of pairs from which the document ids will be collected is *AB, CD* and also *DE*. As there is the need for one more pair to include keyword *E* in this case, the document ids are collected from the last pair which is *DE*. If the keyword pair does not exist in the GBI, then the search procedure stops as there is no possibility of a common document. If all the keywords are connected, then the SvS algorithm is implemented to find the intersection between the keyword pairs to find the common document ids. The SvS algorithm takes the smallest postings list among the search keywords and checks whether the document ids in that list exist in other postings list of the remaining search keywords. A document id in the smallest postings list is kept only if it occurs in all search keyword's postings lists. The SvS algorithm is needed only if there are more than two search keywords. If there are only two search keywords, then the result is directly obtained as it is already indexed in pairs. If all the keywords are connected, then the number of intersections to be made using SvS depends on the number of keywords. In general, if there are *n* search keywords, and $n > 2$ and *n* is even, then only *n/2 - 1* intersections are needed. If $n > 2$ and *n* is odd, then only $\lfloor$ n/2 $\rfloor$ intersections are needed. If $n = 2$, then no intersection operation is needed at all. The algorithm for executing the Boolean AND queries with the inverted index uses the same SvS algorithm except for the fact that GBI implements SvS between keyword pair(s) whereas the inverted index technique implements it between individual keywords in the search keywords set. An important fact to note is that in GBI, the Boolean AND search algorithm stops before executing SvS when there is no relationship between any of the search keywords. In the inverted index, there should be at least one intersection to be performed to validate that no common document exists between the search keywords. For the inverted index, the number of intersections performed using the SvS algorithm is *n-1* for *n* keywords, and with $n > 1$ and all *n* keywords appear together in at least one document. Thus, the number of intersection operations needed in GBI is always lesser than the inverted index.

## III. Experiments and Performance analysis

A proof of concept prototype is built for both GBI and inverted index technique using the PHP scripting language version 7.3.6. This section focuses on the experiments conducted to evaluate GBI in terms of indexing time and Boolean AND queries search time in comparison to the inverted index. Hash table is used to implement both the indices. The experiments are performed on the computing cluster provided by Compute Canada using 1 core of Intel E5-2683 v4 Broadwell CPU and 80 GB memory. The goal is to study the impact of the relationship among search keywords on the performance of GBI for the Boolean AND queries executions. The set of keywords that a particular document can contain is selected beforehand from a given keyword pool to control how the multiple keywords can co-exist in a document. Each experiment was repeated 10 times and the resulting values were averaged.

### A. Workload Generation

The parameters involved in generating synthetic workloads for experiments are discussed next. $D_{count}$ is the total number of documents created and indexed (value ranging from 2 to 10 Million), $K_{pool}$ contains a set of unique keywords which are used to create a document to index in GBI and inverted index and $SK_{count}$ is the number of search keywords. There can be *No*, *Partial* or *Full* relationship between the set of search keywords. A *No* relationship between keywords is a situation in which none of the search keywords have a single common document. A *Partial* relationship between keywords is a situation in which some of the search keywords in the entire search keyword set have some common documents. A *Full* relationship between keywords is a situation in which all of the search keywords in the set have at least one common document. In order to perform experiments for these three relationships, $K_{pool}$ size and $SK_{count}$ are fixed to four. Thus, $K_{pool}$ and the search keywords used in Boolean AND queries contain the same four unique names. The default size of $SK_{count}$ for these experiments is set to four as most Google search queries use less than four keywords [1]. The way the four keywords are utilized to create documents depends on the relationship. In the experiment for the *No* relationship, the total number of documents to index ($D_{count}$) is divided into four equal parts such that documents in each part contain only one of the four search keyword. In the case of *Partial* relationship, the total number of documents to index is divided into two equal parts such that documents in each part contain two of the four search keywords. In the case of *Full* relationship, the total number of documents to index is divided into two parts. Each document in the first part contains all the search keywords and the second part contains only two keywords (out of four).

### B. Experiment Results

The performance metrics monitored in these experiments are described next. $T_s$ is the latency (in milliseconds) associated with the Boolean AND queries evaluation. $T_i$ is the latency (in seconds) and $M_i$ is the memory consumption (in megabytes) associated with indexing a set of documents. Figure 2 shows the performance of the GBI and inverted index techniques when the search keywords are subjected to *No*, *Partial* and *Full* relationships. At first glance, it can be seen that GBI shows a significant time reduction over the inverted index. For example, in the *No* and *Partial* relationship cases, the search time remains constant at approximately 0.02 and 0.023 ms respectively (compared to a linear increase in the case of the inverted index). This is due to the fact that GBI checks whether all the search keywords occur together in any of the documents before performing an intersection. Checking for the simultaneous occurrence of two keywords in a document is a simple

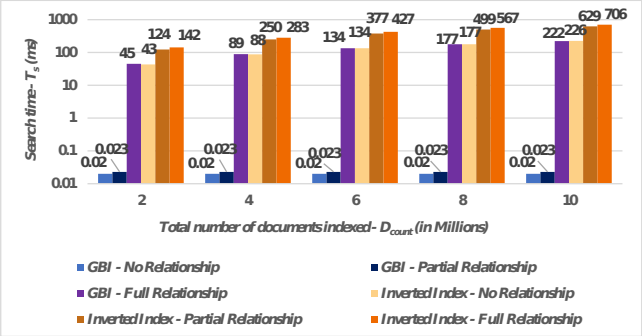


Fig. 2. Search latency ($T_s$) for Boolean AND queries with respect to *No*, *Partial* and *Full* relationship between keywords in GBI and inverted index.

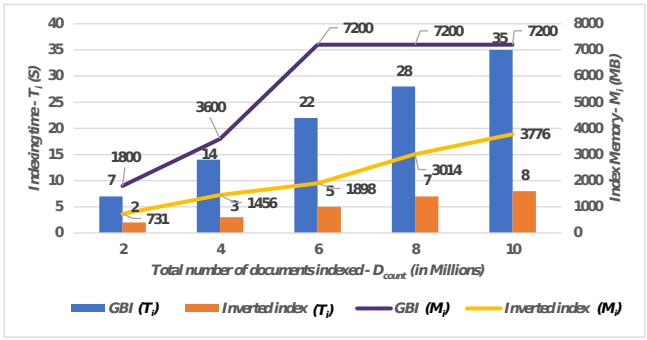


Fig. 3. Indexing time ($T_i$) and memory consumption ($M_i$) for GBI and inverted index

lookup for an entry represented by those two keywords in the GBI hash table. Since there is no simultaneous occurrence of all the search keywords in case of *No* relationship and no simultaneous occurrence of some keywords in the experiments for *Partial* relationship, the GBI stops further action and does not perform any intersection operation. In the case of *Full* relationship, the intersection in GBI is done in pairs and so in this experiment, the number of intersections performed is only one as opposed to three intersections performed with inverted index. Thus, GBI is faster in all the three cases with overall improvements of 99.9% for *No* and *Partial* relationships and 69% for *Full* relationship. In the inverted index, the execution time for Boolean AND queries depends on the size of the smallest postings list among the search keywords. As the total number of documents $D_{count}$ indexed increases, the size of the smallest postings list also increases leading to an increase in the execution time for the Boolean AND queries. In conclusion, we can see that as keywords are more and more related in any number of documents, the Boolean AND search time suffers.

To obtain a broader view about the time ($T_i$) and memory ($M_i$) consumed by GBI and inverted index, a set of 2 to 10 Million documents in steps of 2M are created. Each document in the set is created by randomly choosing at least two unique keywords from a pool of 10 keywords. Then, Algorithm 1 is used to index the generated documents. Figure 3 shows that inverted index results in smaller $T_i$ and $M_i$ in comparison to GBI. This is because, in the inverted index, each keyword is stored individually with its postings list. However, in the case of GBI, there will be a postings list for every keyword pair which consumes more time and memory to create. To accommodate more data and to avoid resizing often, the hash table increases its size in powers of two[3]. This explains the significant memory difference between GBI and inverted index when $D_{count}$ is 6M and also why $M_i$ is constant from 6M to 10M in GBI. Although GBI can perform search operations faster, more time and more memory are required for the initial indexing of documents.

[3]https://nikic.github.io/2014/12/22/PHPs-new-hashtable-implementation.html

## IV. Conclusions

Indexing data for effective searching is a challenging problem. In this paper, a new graph-based indexing technique is proposed to effectively evaluate Boolean AND queries that are often used by search engines like Google. Compared to the inverted index, the performance enhancement achieved with GBI is significantly higher (99.9%) when search keywords do not appear together in any documents. Similarly, GBI shows an improvement of 69% when all the search keywords appear together in one or more documents for the different range of values for $D_{count}$. Due to high indexing time of GBI when compared to the inverted index, GBI could provide substantial benefits in applications where indexing is performed only once and search is performed multiple times.

Directions for further research include: 1) Investigating methods for reducing indexing time and memory for GBI. 2) Extending GBI to evaluate other types of queries such as Boolean NOT and OR. 3) Comparing GBI with an enterprise-level product such as Lucene that uses an inverted index.

## Acknowledgment

We are thankful for the computing resource support provided by Compute Canada. We are also grateful to TELUS and NSERC of Canada for providing financial support for this research.

## References

[1] W. Stream, "Google search statistic," Mar. 2020. [Online]. Available: https://www.wordstream.com/blog/ws/2019/02/07/google-search-statistics

[2] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[3] A. K. Mahapatra and S. Biswa, "Inverted indexes: Types and techniques," *International Journal of Computer Science Issues*, vol. 8, no. 1, pp. 384–392, July 2011.

[4] R. Sharma, A. Kandpal, P. Bhakuni, R. Chauhan, R. H. Goudar, and A. Tyagi, "Web page indexing through page ranking for effective semantic search," in *2013 7th International Conference on Intelligent Systems and Control (ISCO)*, Jan 2013, pp. 389–392.

[5] S. Pohl, A. Moffat, and J. Zobel, "Efficient extended boolean retrieval," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 1014–1024, June 2012.

[6] J. S. Culpepper and A. Moffat, "Efficient set intersection for inverted indexing," *ACM Trans. Inf. Syst.*, vol. 29, no. 1, pp. 1:1–1:25, Dec. 2010. [Online]. Available: http://doi.acm.org.proxy.library.carleton.ca/10.1145/1877766.1877767