Static Term Result Caching in Term-based Partition

Derin Karadeniz
Department of Computer Engineering
Middle East Technical University
06800, Ankara, Turkiye
derin.karadeniz@metu.edu.tr

Yusuf Mirza Altay
Department of Computer Engineering
Middle East Technical University
06800, Ankara, Turkiye
mirza.altay@protonmail.com

ABSTRACT

The retrieval of information from large-scale databases is a crucial task in modern search engines. One of the key components of search engine performance is the caching of query results, which allows the system to answer the user in shorter durations. The cache can be constructed through the use of several strategies such as term frequency (TF) and term frequency stability (TFS) combined with term frequency caching methods. In this research, we investigate the differences between TFS caching by implementing a term-based partitioning in a search engine. Our experiments show that while TFS caching is expected to show high performance over TF caching, TS caching can lead to poor performance when it is used within term-based partitioning over a small dataset. We provide insights into the relative effectiveness of these caching methods for improving search engine performance.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – Search process; H.3.4 [**Information Storage and Retrieval**]: Systems and Software – Distributed systems, Performance evaluation (efficiency and effectiveness)

General Terms

Performance, Experimentation

Keywords

Caching, Search Engine, Term-based Partition, Term Caching, List Caching

1. INTRODUCTION

Caching plays a vital role in the performance of search engines, as it enables faster retrieval of results and reduces the load on the index. The concept of caching in search engines dates back to the early days of information retrieval, where caching of frequently accessed documents was used to improve retrieval speed.

With the advent of the World Wide Web and the explosion of online information, the need for efficient caching methods in search engines became even more pressing. One of the earliest forms of caching used in search engines is posting list caching with the score function results, which involves caching the posting lists of terms in the nodes of the index.

In recent years, TF caching is used widely. TF caching is based on the idea that frequently occurring terms in a query are more likely to be relevant to the user than infrequent terms. Therefore, search engines using TF caching prioritize the caching of terms that occur frequently in queries.

On the other hand, TFS caching is based on the idea that stable terms, or terms that are consistently present across multiple queries, are more likely to be relevant to the user than unstable terms. Therefore, search engines using TS caching prioritize the caching of terms that are consistently present across multiple queries.

This research paper aims to investigate the differences between TF and TFS caching on a search engine that uses a term-based partitioning approach. The term-based partitioning approach involves processing each term on a previously assigned node, rather than processing a term of a query in multiple nodes which contains the documents related to the term. This allows for a more detailed analysis of the impact of each caching technique on search engine performance, including cache hit ratios. The results of this study will provide insight into the best caching technique to use in a search engine and inform future research in this area.

2. RELATED WORK

Static caching methods have been widely studied in the context of search engines to improve query processing efficiency. One such method is caching the posting lists of query terms. Long and Suel [1] proposed a three-level caching approach for large web search engines. The approach utilizes a combination of memory-based,

disk-based, and remote caching to reduce the number of disk accesses and network transmissions. The authors conducted experiments on a large web search engine dataset and showed that the proposed approach achieved a significant reduction in query processing time.

Baeza-Yates et al [2] studied the impact of caching on search engines. They proposed a caching strategy that considered the query popularity, recency, and freshness of the query results. The authors conducted experiments on a real-world search engine dataset and showed that the proposed strategy improved the search engine's performance by reducing the number of server accesses and network transmissions.

Ozcan et al [3] proposed a static query result caching strategy that considered both the cost of recomputing the query results and the cost of storing the results in the cache, as well as the query frequency and query frequency stability. The authors proposed a cost model that takes into account the query frequency and stability, which is defined as the degree of change in the query frequency over time. The authors conducted experiments on a real-world search engine dataset and showed that by considering the stability of query frequency, their proposed strategy improved the search engine's performance by reducing the number of server accesses and network transmissions while also achieving a high cache hit rate. The stability of the query frequency was found to be a crucial factor in determining which queries should be cached and for how long. The authors also found that the proposed strategy was able to adapt to changes in the query frequency over time, resulting in a more efficient use of the cache.

In [4], Ozcan et al proposed a cost-aware caching strategy for query result caching in web search engines. The authors proposed a cost model that considers the costs of recomputing the query results and the costs of storing the results in the cache. The authors conducted experiments on a real-world search engine dataset and showed that the proposed strategy improved the search engine's performance by reducing the number of server accesses and network transmissions.

In summary, previous research has focused on static caching methods such as caching posting lists of query terms and query results. These methods have been shown to improve the efficiency of query processing in web search engines by reducing the number of server accesses and network transmissions.

3. TERM FEATURES

To select terms into the static cache, we used the below term features in the experiments.

- **Term Frequency (TF)**: The term frequency is a parameter for the most frequent terms in the training query log.
- **Term Frequency Stability (TFS)**: The frequency stability first introduced by Ozcan et al [1] over the query frequency stabilities. Term frequency stability is inspired by that research.

  This feature calculates the differences of a term frequency between intervals when the training query log is divided into a dedicated number of intervals.

4. METHOD

As indicated previously, the caching type that has been implemented in this paper is static. Static caching is a technique used to store the output of a process in a storage area, such as RAM, which provides relatively faster access when compared to a hard disk or a flash region. This type of caching has several application areas such as dynamic web pages, cloud computing, or, in this case, a high-complexity operation at a search engine’s server, so that the result can be quickly retrieved the next time the same page or operation is requested. This can significantly improve the performance of search engines, web pages, and applications, as the cached version can be served to the user much more quickly than the non-cached version.

There are different strategies to build up a static cache. In this paper, two of them have been focused on and compared: frequency and frequency stability of cached items. This build process requires analyzing data collection. So, a recent set of queries or term logs should be preferred in order to increase the cache accuracy.

Frequency-based strategy is an intuitive one. At its simplest, it seeks the occurrence of the item in a certain period, then puts the items in order by their number of occurrences and chooses the top items to construct the static cache.

Stability-based strategy [3] is a slightly more complex method that considers the variation in the appearance frequency of the item. In this paper, the algorithm indicated in [3] was altered for the search engine model in use.

4.1 Search Engine Model

Before talking about the details of the stability-based method, it would be better to describe the structure of the search engine model since the original algorithm is accordingly modified. The search engine model that has been used, consists of a broker and multiple nodes which have been partitioned in term-based manner. In term-based partitioning, each term is assigned to a specific node and its posting list is stored in the node's storage. Therefore, when a query is received by the broker, it pre-processes the query to extract the terms and sends them only to the corresponding nodes which contain the query's terms. As a result, solely, particular nodes will work.

The reference paper [3] applies the stability feature to create a query cache, at the broker, which stores term and top-K result pairs. So, when a query is in the cache, its final results have been already calculated and can be returned quickly to the user without executing lower-level operations on nodes.

Nevertheless, in this article, the term frequency, and term frequency stability features were integrated into the partition level cache. In this cache, terms are stored within an array containing IDs of the documents in the terms posting list alongside the documents' calculated BM25 scores. So that, whenever a term is sent by the broker to the node if the term exists in the node's cache, the best-scored documents are directly delivered to the broker for top-k result calculation with other delivered term results. Otherwise, the term's posting list should be read from the disk or flash and then processed to determine the BM25 scores of each document. This sequence of operations would take much longer time than the cached terms'.

4.2 Term Frequency Stability (TFS)

To create a static cache, the term frequency stability approach parses timestamped query logs to terms by tokenizing them and divides them into N equal time intervals by their timestamp. Then, it determines the term's frequency at each interval. After that, the mean interval frequency is calculated. Finally, the deviation of each interval's frequency from the mean interval frequency is calculated and summed up. To normalize the obtained sum, it is divided by the mean interval frequency. The resulting stability scores are directly used to choose the terms which will be inserted in the node's cache. Assuming N nodes, this algorithm will be used in all N nodes and create N separate caches dedicated to the nodes themselves. The equation that is previously described is the same as the QFS equation indicated in [3]. It is defined by following formula:

$$TFS_i = \sum_{j=i}^{n} \left| f_{ij} - f_m \right| / f_m$$

where i is the index of a term, j is the index of a time interval. This means f_{ij} is the $term_i$ frequency in the $time\ interval_j$, and f_m is the mean of the $term_i$ frequency across training query log.

5. EXPERIMENTAL RESULTS

5.1 Comparing Intervals

The first experiment shows the differences between average cache hit rates by setting different intervals for TFS. In this experiment, TFS is used only. The training data belongs to a query log across 11,2 hours which is the period that we divide into intervals.

The node count is set to 32 across all experiments. For every node cache size is set to 10000 posting list which has document ID, and BM25 scores of that document. Moreover, for every term in the cache, the max posting list size is set to 300. In other words, every term has at most 300 document ID, BM25 scores pairs.

Since we set a limit for the number of the document ID and BM25 score pairs, some nodes might have more terms than other nodes which might lead to higher cache hit ratios since we are using term-based partitioning.

Table 1 shows the impact of the time interval numbers. The cache hit ratio increases with the interval number when using TFS as the caching method for terms.

Table 1: Average Hit Ratios (%) vs. Number of Time Intervals

Number of Time Intervals	TFS
2	0,4866
4	0,5046
8	0,5074
16	0,5125

32	0,5172
64	0,5228
128	0,5243

5.2 Comparing Term Frequency And Term Frequency Stability

The second experiment compares the TF over TFS hit ratios with varying cache sizes. The remaining setup is the same as the first experiment. The number of time intervals in this experiment is chosen 64.

The number is chosen as 64 because

Table 2 results show that TF generally gives better cache hit ratios than using TFS when used with relatively small cache sizes. Moreover, when the cache size becomes larger the hit ratios of TFS catch the TF hit ratios.

Table 2: Average Hit Ratios (%) vs. Cache Sizes

Cache Size	TF	TFS
5K	0,3999	0,3993
10K	0,5247	0,5228
15K	0,6014	0,6002
20K	0,6567	0,6557
40K	0,7878	0,7863
100K	0,9064	0,9064
200K	0,9064	0,9064

6. CONCLUSION

In this research, we showed the differences of average cache hit ratios between TF and TFS. In all of the experiments, results showed that TF gives slightly better results than using TFS for caching terms. Even though some nodes might get higher hit rates when comparing TFS to TF hit rates one by one, the overall result does not change. Moreover, TFS cache hit ratio gets better when dividing the query log into a higher number of time intervals.

7. ACKNOWLEDGMENTS

This research is done under the METU Computer Engineering Department's Web Search Engine Design class with the lead of Ismail Sengor Altingovde.

8. REFERENCES

[1] Long, X., Suel, T. Three-Level Caching for Efficient Query Processing in Large Web Search Engines. World Wide Web 9, 369–395 (2006). https://doi.org/10.1007/s11280-006-0221-0

[2] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. 2007. The impact of caching on search engines. In Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '07). Association for Computing Machinery, New York, NY, USA, 183–190. https://doi.org/10.1145/1277741.1277775

[3] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. 2008. Static query result caching revisited. In Proceedings of the 17th international conference on World Wide Web (WWW '08). Association for Computing Machinery, New York, NY, USA, 1169–1170. https://doi.org/10.1145/1367497.1367710

[4] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. 2011. Cost-Aware Strategies for Query Result Caching in Web Search Engines. ACM Trans. Web 5, 2, Article 9 (May 2011), 25 pages. https://doi.org/10.1145/1961659.1961663