An Intersection Cache Based on Frequent Itemset Mining in Large Scale Search Engines

Wanwan Zhou, Ruixuan Li, Xinhua Dong
School of Computer Science and Technology
Huazhong University of Science and Technology
Wuhan, P.R.China
E-mail: {zhouwanwan, rxli, xhdong}@hust.edu.cn

Zhiyong Xu
Department of Mathematics and Computer Science
Suffolk University
Boston, USA
E-mail: zxu@suffolk.edu

Weijun Xiao
Department of Electrical and Computer Engineering
Virginia Commonwealth University
Richmond, USA
E-mail: wxiao@vcu.edu

***Abstract*—Caching is an effective optimization in large scale web search engines, which is to reduce the underlying I/O burden of storage systems as far as possible by leveraging cache localities. Result cache and posting list cache are popular used approaches. However, they cannot perform well with long queries. The policies used in intersection cache are inefficient with poor flexibility for different applications. In this paper, we analyze the characteristics of query term intersections in typical search engines, and present a novel three-level cache architecture, called TLMCA, which combines the intersection cache, result cache, and posting list cache in memory. In TLMCA, we introduce an intersection cache data selection policy based on the Top-N frequent itemset mining, and design an intersection cache data replacement policy based on incremental frequent itemset mining. The experimental results demonstrate that the proposed intersection cache selection and replacement policies used in TLMCA can improve the retrieval performance by up to 27% compared to the two-level cache.**

Keywords—search engine; cache; intersection cache; frequent itemset mining.

I. INTRODUCTION

With the explosive growth of data on the Internet, efficient data storage and retrieval strategies are becoming more and more important for search engines since they are one of the most important applications on the Internet. Hard disk is used as the major media to store the massive data for many large scale search engines. The low disk I/O access speed has become the major bottleneck for data retrieval operations. Data caching mechanism can effectively improve the retrieval performance. Different caching algorithms have recently been developed, such as result cache, intersection cache, projection cache, posting list cache, snippet cache, and document cache.

Result cache and posting list cache are the most studied caches so far. However, result cache only performs very well on single-term and two-term queries while posting list cache needs extra calculations to return the final result. Intersection cache happens to play a complementary role for the previous two caches. However, in practice, the number of terms is considerable numerous, the combinations of multiple terms are tremendously huge. It becomes very difficult to choose which intersection should be kept in the cache. Thus, the existing intersection cache cannot achieve the satisfactory caching performance. It consumes a large amount of disk space to keep the information and only a very low hit rate can be reached. Furthermore, the low speed disk I/O accesses also reduce its effectiveness. In this paper, we design a novel 3-layer cache architecture called TLMCA to address this issue through keeping the most suitable intersection data in the memory to improve retrieval performance. Especially for longer queries, intersection cache could hit high-frequency term combinations, which makes up for result cache. Meanwhile, it saves inverted lists intersection computation, which makes up for posting list cache. In addition, we propose new intersection cache selection and replacement policies, which include intersection cache data selection policy based on Top-N frequent itemset mining (FIMI) and intersection cache data replacement policy based on incremental frequent itemset mining (IFIMI). To the best of our knowledge, our work is the first to integrate frequent itemset mining technology into the intersection cache.

The contributions of this paper are as follows.

- First, we analyze the characteristics of query term combinations that lay foundation for cache strategies.
- Second, we propose TLMCA, a novel three-level cache architecture, and integrate frequent itemset mining algorithms with intersection cache to improve the efficiency and flexibility.
- Third, we conduct extensive simulation experiments to evaluate TLMCA performance. The results show significant benefits of retrieval performance for TLMCA compared to other strategies.

The rest of the paper is organized as follows. Section II discusses the related work. Section III analyzes the characteristics of dataset and query log in large scale search engines. Section IV describes the system design of TLMCA, the three-level cache architecture. Section V presents the intersection cache data selection and replacement policies based on frequent itemset mining. Section VI demonstrates the experimental results. Finally, Section VII concludes the paper and briefly discusses the future research directions.

II. Related work

A. Search Engine Cache

The most common cache in search engine is the single level cache, which can be further categorized as result cache, score cache, intersection cache, projection cache, posting list cache, snippet cache, and document cache.

1) Result cache (RC): RC preserves the most frequently queried results in the cache. Its management is simple. However, it is coarse-grained, and its hit rate decreases very fast as the data volume increases.

Markatos [1] analyzed the EXCITE query log, and showed that static query result caching is a good choice only for small cache sizes, while dynamic caching is better for large cache sizes. Fagni [2] proposed an SDC policy that divides the cache into two parts and one part is reserved for static caching and the other one is used for dynamic caching. Ozcan et al. [3] introduced a model based on the stability of frequency and cost-aware result caching policies. Similarly, Gan et al. [4] proposed and evaluated a set of feature-based result cache eviction policies to achieve significant improvements. Wang et al. [5] analyzed several result cache policies of search engine. Recently, researchers are focusing on the update strategy in result cache to improve the performance [6, 7]. Namely, the retrieval system should always return the latest results to the user in dynamic environment.

2) Posting list cache (PLC): The hit ratio in PLC is relatively higher than result cache because it uses a fine-grained approach. However, it needs substantial calculation to return the final result, and its management is more complex.

Zhang et al. [8] evaluated several state-of-the-art inverted list compression methods and different list caching policies. Baeza-Yates et al. [9,] proposed a new algorithm for static caching of posting lists that outperforms previous static caching algorithms as well as dynamic algorithms. Various posting list selection and replacement policies, such as LRU, LFU, a strategy based on frequency and the ratio of frequency to the size (FreqSize), have been developed. Currently, the research focuses on inverted index are index structure, clipping and compression algorithms.

3) Intersection cache (IC): IC is the intersection of posting lists of several terms that appear together in one query. It can improve the retrieval performance and can be applied as a complementary method for the aforementioned two caches. A drawback is that the amount of terms' combinations could be too large to keep in the memory. Thus, it has to consume a lot of disk space. Therefore, it is difficult to choose which intersection data should be stored in the cache.

Long et al. [10] proposed a three-level cache system. They placed the projection cache on top, and use the hard disk as the basis of the two-level cache (RC and PLC). Ozcan et al. [11] proposed a five-level static cache architecture. Feuerstein et al. [12] proposed and evaluated static, dynamic cost-aware policies and hybrid policies for IC with inverted index residing on disk and in main memory. Later, they proposed and evaluated a static cache that works simultaneously as list and intersection cache [13]. Wang et al. [14] found that posting list intersection is the bottleneck in SSD-based search engines and exploited full-term-ranking-cache (FTRC) and two-term-intersection-cache (TTIC) to mitigate list intersection overhead.

There are also some researches on snippet caching [15] and document caching [16]. Some studies aimed to integrate multiple types of caches [10, 11]. In this paper, we focus on the intersection cache, and will discuss it in the next section.

B. Frequent itemset mining

The most popular frequent itemset mining (FIMI) algorithms are Apriori and FP-Growth. The Apriori algorithm was proposed by Agrawal [17] in 1994. However, it has to scan the data set repeatedly and produces a large number of candidate itemsets. Han et al. [18] proposed FP-Growth which does not generate candidate frequent itemsets. It only needs to scan the database twice and avoids the generation of a large number of candidate itemsets. Various algorithms such as maximal FIMI, closed FIMI, Top-N FIMI, incremental FIMI were developed based on Apriori and FP-Growth.

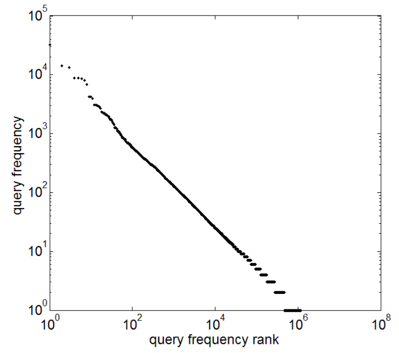
Top-N FIMI algorithm [19,20] needs to set the number of frequent itemsets N rather than the minimum support. It produces a backtracking problem so that the space and time complexity becomes very high. Traditional FIMI algorithms are based on the static data sets. However, data sets in reality mostly are dynamic, which has addition, deletion and modification, and they lead to some existing frequent itemsets become invalid and some other previous non-frequent itemsets turn to be frequent. Completely mining again towards the new data will bring too much overhead. Thus, incremental FIMI algorithms arise. They always divide the dynamic data set into landmark window, attenuation window and sliding window for processing. The data structures, such as bit table, binary vector, matrix and the prefix tree, are often used.

In summary, the result cache and posting list cache cannot perform well for multi-term queries. The previous intersection cache selection policies are not very efficient, and they only consider term pairs. Our objective is to explore a simple and efficient intersection cache solution with good retrieval performance.

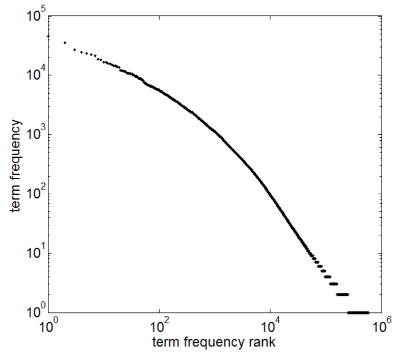
III. Query log analysis

The dataset used in our experiments is an English data set enwiki downloaded from Wikipedia with about 5 million documents included. The compression index file is 5.2GB. The query log comes from the AOL search engine, containing approximately 3519003 queries in total. Among them, there are 1197567 different queries, 580116 different query terms, and 6535327 different intersections. The average length of a query is 2.23, and the longest query contains 132 terms. We analyze the search engine query log, and find that it has some obvious query term combination characteristics.

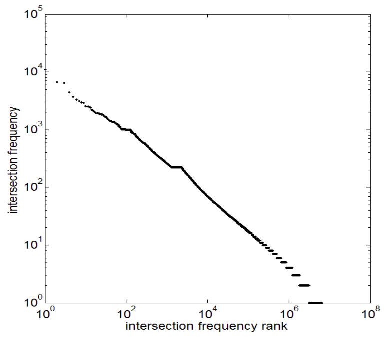
(1) As shown in Fig. 1, the query, query term, and intersection all follow the power-law distribution with α being 0.73, 1.1, and 0.76, respectively. (2) The majority of the queries have a length less than or equal to 3. However, queries need to occupy 80% of the total workload in search engine whose length is more than 3 [6]. (3) With the increase of k, the



(a) Query frequency distribution



(b) Term frequency distribution



(c) Intersection frequency distribution

Fig. 1. Frequency distribution of the query data set

support of *k*-itemset is in non-ascending order. (4) The number of hit counts of the intersection is relatively low, which is one or two in most cases. (5) Users' queries meet the closure properties of frequent itemsets.

The result cache and posting list cache make good use of the query's and term's localities. They have very good performance on queries whose length is less than 3. Nevertheless, queries whose length is larger than 3 often need to consume more I/O and computing resources. From the above discussions, we can observe that the intersection cache can be used to alleviate this problem. Thus, in our TLMCA design, we integrate all of them together to achieve optimal performance.

IV. THREE-LEVEL CACHE ARCHITECTURE

In TLMCA design, we add an intersection cache on top of the common two-level cache (RC and PLC) in the memory. The caching system architecture is shown in Fig. 2.

As described in the previous section, in TLMCA, result cache and posting list cache adopt the relatively simple selection policy based on frequency, and the commonly used replacement policy of result cache and posting list cache, such as LRU, LFU, FreqSize etc, can be applied. However, the intersection cache data selection policy is designed based on the Top-N frequent itemset mining. We propose a new replacement policy with the incremental frequent itemset mining based on sliding window to deal with dynamic intersection cache, which will be introduced in detail shortly.

When the search engine receives a user's query with *n* terms (t_1, t_2, … , t_n), the first query process step s1 is to detect the result cache. The second step s2 is to check intersection cache. The system extracts all k-itemsets ($2\leq k \leq m$, m is the maximum length of the IC), matches the intersection cache from the longest to the shortest depending on the length, and returns an intersection's posting list immediately when hitting an item, say I(t_1, t_2, t_3). The third step s3 is to review the posting list cache with the last remaining terms (t_4, t_5, …, t_n). It will take out term t_4's posting list from the cache, and fetch (PL(t_5), …, PL(t_n)) by accessing the hard disk at last. The fourth step s4 is that Index Servers return the result (r_1, r_2, ⋯, r_k) to the Web Server after the operations of intersection (I(t_1, t_2, t_3)∩PL(t_4)∩PL(t_5)∩⋯∩PL(t_n)), page ranking and snippet generation and so on. The Web Server receives partial results from each Index Server. Finally, the fifth step s5 is to generate a new top-k query result (r_1', r_2', ... , r_k') after summary, forms a final result page, and returns it to the user.

V. AN INTERSECTION CACHE DATA STRATEGY BASED ON FREQUENT ITEMSET MINING

In TLMCA, we design a Top-N frequent itemset mining algorithm based on FP-Growth for our purpose. It treats the query log as a transaction database D, terms are considered as the items in D. We introduce several parameters, including the number of frequent itemset mining (N=300,000) and the maximum length of the intersection cache (maxLength=3). Algorithm 1 shows the detail procedure of intersection cache selection strategy based on the frequent itemset mining (ICSS_FIMI).

Algorithm 1. ICSS_FIMI

```
1  Input: maxLength, N, query log on hard disk
2  Output: a list of Top-N intersections
3  while (the term in the query log)
4     Calculate the support of the term
5  end while
6  Sort terms in descending order according to their support
7  while (the query in the query log)
8     Reorder terms in descend according their support
9  end while
10Build a complete frequent prefix tree (FP-Tree)
11 using the processed queries
12 while (the term pair in the query log)
13    Calculate the support of the term pair
14 end while
15 Sort term pairs in descending order according to
16    their support
17 $Sup_{min}$ = the support of the Nth term pair
18 while (node on the FP-Tree)
19     if ($Sup_{min}$>the support of the node)
20         Delete the node from the FP-Tree
21     end if
22 end while
23 Create a list
24 while(node on the FP-Tree)
25     Build the frequent conditional pattern base (FCPB)
```

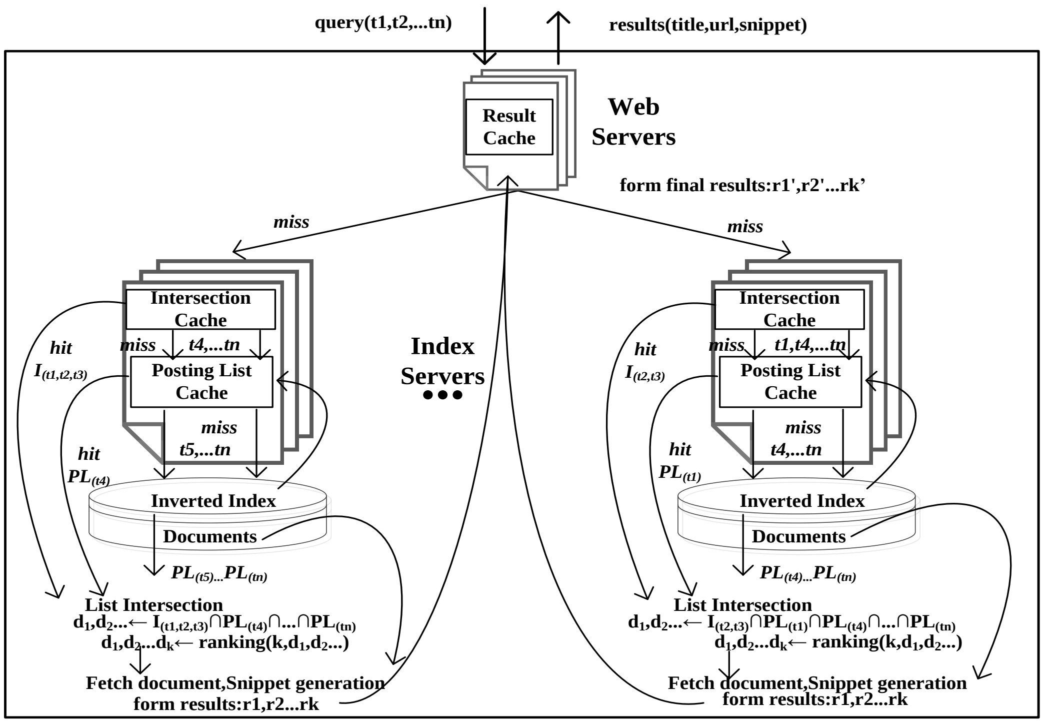


Fig. 2. Caching system architecture in search engine

```
      of the node
   **if**(2<=the size of intersection<=*maxLength*)
      **if**(the support of the intersection >= $Sup_{min}$)
            Add it and its support to the list
      **end if**
   **end if**
**end while**
Sort all the intersections in the list according to their
   support
**while**(the intersection in the list)
   **if**($N$<=the intersection's index)
      Remove the intersection
   **end if**
**end while**
**return** L
```

As shown in Algorithm 1, it mainly takes advantage of the third characteristic of the query log. The space and time overhead is relatively low when using FP-Growth algorithm to mine out the Top-N frequent itemsets. The algorithm's time complexity is $F=O(M^2*\log M + M*T)$. Since the time overhead mainly depends on term pairs' sorting and searching the condition pattern base, where M is the number of different query term and T is the number of query. In TLMCA, we filter out queries whose length is too long because these queries are seldom, they will seriously affect the performance of the algorithm but have little improvement on the intersection data to be mined out.

In TLMCA, we adopt a periodical global replacement strategy for intersection cache to maintain intersection cache data's time-effectiveness. However, the data structure of FP-Tree strongly depends on the support of each item and the minimum support. It leads to the release and rebuilding of FP-Tree in memory and re-mine out frequent patterns again. Therefore, we introduce the Trie-Tree data structure to provide efficient intersection cache data replacement strategy based on incremental frequent itemsets mining (ICRS_IFIMI).

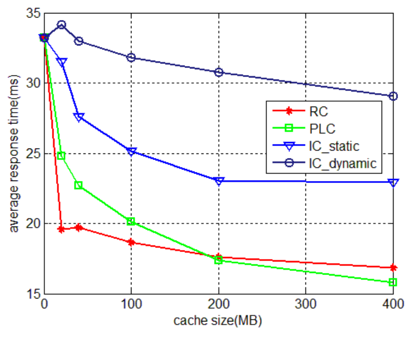
VI. Performance evaluation

TABLE 1 summarizes the hardware and software environment settings. Our simulative search engine is based on Lucene3.0.0. For simplicity, we use one single search node instead of distributed search cluster. In our experiments, we use the last one hundred thousand queries as test data and the previous queries as training data. Our experiments test with AND query semantics and other logical operators will be added in the future. We test with longer data set in the experiments, and the result is similar. Our evaluation compares the retrieval performance of result cache and posting list cache, and examines the efficiency of intersection cache data strategies.

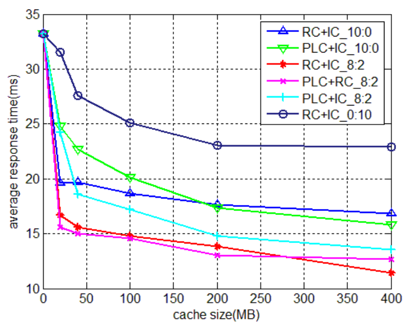
A. *Retrieval Performance with Different Architecture Levels*

The retrieval performance comparison and analysis are among one-level cache (1LC), two-level cache (2LC) and three-level cache (3LC). At the same time, we also compare the response time of existing intersection caching policy, and explore the characteristics of intersection cache data itself. The memory cache size ranges from 0 to 400MB in all experiments.

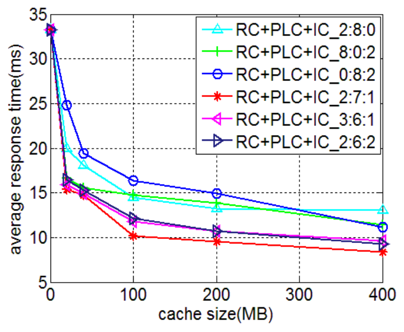
1) One-level Cache: Fig. 3(a) shows the average response time of the one-level cache. "RC" and "PLC" are the dynamic result cache and the posting list cache, "IC_static" is the static intersection cache and "IC_dynamic" is the dynamic intersection



(a) The performance of 1LC

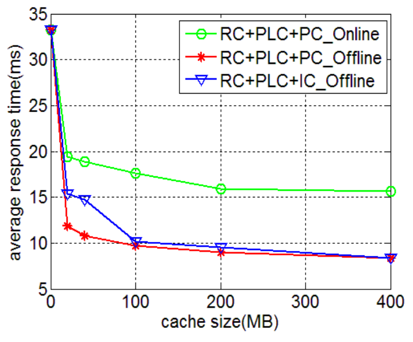


(b) The performance of 2LC

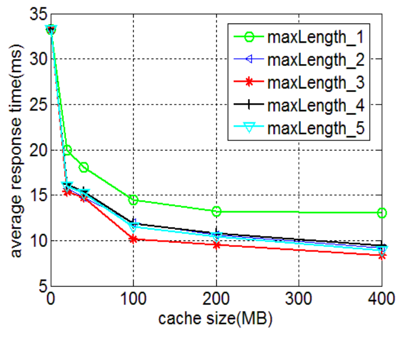


(c) The performance of 3LC

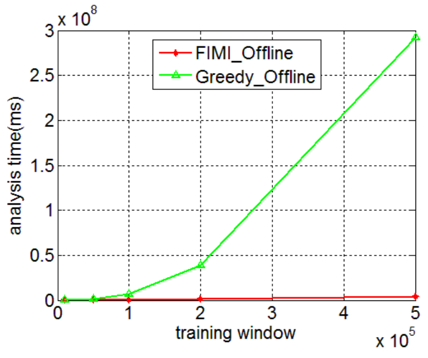
Fig. 3. Retrieval performance with different architecture levels



(a) The performance of different strategies



(b) The performance of different makLengths



(c) Selection efficiency of intersection cache

Fig. 4. Performance comparison over different strageties

TABLE 1. HARDWARE AND SOFTWARE ENVIRONMENT SETTINGS

Test-plat form Environment	
IR Tool	Lucene 3.0.0
Data set	Enwiki-20090805-pages-articles.xml
Query log	AOL-user-ct-collection
CPU/RAM	Intel Core2DuoP8600(2.40GHz\1066MHz\3072KB)/4GB
OS	Window 7/ Ubuntu 10.04
HDD	HITACHI HTS545025B9A300 250GB

cache. All the dynamic caches apply the simple LRU replacement policies. As we can see from the figure, the dynamic IC performs the worst. IC itself can bring retrieval performance improvement to some extent. However, it is not as prominent as the RC and PLC.

2) Two-level Cache: Fig. 3(b) shows that the retrieval performance of two-level cache. “RC+IC_10:0” means that only result cache occupies the memory, and “PLC+RC_8:2” means that the proportion of memory space allocation for posting list cache and result cache is 8:2. We can observe that all two level cache policies perform better than one-level cache. From a series of experiments, the retrieval performance of two-level cache is better when the memory space proportion of RC and PLC, RC and IC, PLC and IC are 2:8, 8:2 and 8:2, respectively.

3) Three-level Cache: We use a small amount of memory to store intersection cache besides the result cache and posting list cache, which forms a three-level cache. The result is shown in Fig. 3(c). From the figure we can see that all three-level cache strategies perform better than two-level cache ones. The average performance improvement is 27%. When the memory space proportion of RC, PLC and IC is 2:7:1, the system has the least response time. This is mainly because the small intersection cache hit the frequently accessed query term combinations. It not only reduces the disk I/O accesses, but also reduces the index intersection computation overhead.

B. *Performance Comparison over Different Strategies*

To investigate the performance of different strategies, we carry out experiments to compare with the offline Greedy strategy and the online Landlord strategy proposed in [6]. The result is shown in Fig. 4(a), where “RC+PLC +PC_Online” is the three-level cache of projection online Landlord strategy, “RC+PLC+PC_Offline” is its projection offline Greedy strategy, and “RC+PLC+IC_Offline” is our strategy. As shown in the figure, the online method has the worst performance. The offline Greedy strategy performs much better for small caches. With the increase of the cache capacity, our strategy performs roughly the same as the offline Greedy strategy. However, the projection selection efficiency is very low. We will discuss it in Section VI(C).

On the other hand, we further investigate the optimum maximum length (*maxLength*) of the intersection. The retrieval performance is the best when the maximum length of intersection is 3. The result is shown in Fig. 4(b). The optimal length can be adjusted according to different characteristics of query in our intersection cache data selection strategy. Hence, our strategy is more flexible.

C. Selection Efficiency of Intersection Cache

In this experiment, we study the efficiency of different intersection strategies. Fig. 4(c) shows the efficiency of the offline projection selection and our intersection selection. In the process of offline selection of the intersection data, the offline Greedy strategy needs to consume a tremendous long time for a large amount of training data though the analysis is off-line. The analysis time means the time to mine top three hundred thousand intersections from the training data while the training window ranges from ten thousand to five hundred thousand. The offline Greedy strategy uses about 4 days when the training queries are five hundred thousand. Thus, it is not acceptable in practice. Our FIMI strategy runs much faster, it only needs about 65 minutes due to the adoption of FP-Tree structure. It can improve the time and space overhead in the analysis process. In addition, our strategy is more flexible and space-saving to set the maximal length of the intersection according to the query characteristics.

VII. Conclusion and future work

In this paper, we analyze the search engine query log characteristics and then propose TLMCA, a three-level cache architecture in which we add a static intersection cache on the basis of result cache and posting list cache in the memory. The intersection cache selection strategy is based on the Top-N FIMI of high efficient FP-Growth and the intersection cache replacement strategy is based on the incremental frequent itemset mining of Trie-Tree. The retrieval performance has been improved significantly. With the novel design, TLMCA system not only reduces the number of random disk I/O accesses, but also reduces the CPU computational overhead. Furthermore, we also explore the property of the intersection cache itself and its effect on result cache and posting list cache.

Overall, this paper is the first to use FIMI strategy to select intersection cache data for search engines. There are several issues need further explorations. First, in this work, we assume that the index files stored on HDD are static. However, the dynamic scenario should be considered. Second, along with the gradual rise of in-memory computing in recent years, the optimization of the cache and index structures in memory could be considered to further improve the retrieval performance.

Acknowledgements

This work is supported by National Natural Science Foundation of China under grants 61173170, 61300222, 61433006 and U1401258, Innovation Fund of Huazhong University of Science and Technology under grants 2015TS069 and 2015TS071, and Science and Technology Support Program of Hubei Province under grant 2014BCH270.

References

[1] E P. Markatos. “On caching search engine query results”, Computer Communications, 2nd ed, vol 24, 2001, pp. 137-143.

[2] T. Fagni, R. Perego, and F. Silvestri, “Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data”, ACM Transactions on Information Systems, 1st ed, vol 24, 2006, pp. 51-78.

[3] R. Ozcan, I S. Altingovde, and Ö. Ulusoy, “Cost-aware strategies for query result caching in web search engines”, ACM Transactions on the Web, 2nd ed, vol 5, 2011, pp. 1-25.

[4] Q. Gan, T. Suel, “Improved techniques for result caching in web search engines”, The 18th International Conference on World Wide Web, Madrid, Spain, 2009, pp. 431-440.

[5] J. Wang, E. Lo, M L. Yiu, et al. “The impact of Solid State Drive on Search Engine Cache Management”, The 36th International ACM SIGIR conference on research and development in Information Retrieval, Dublin, 2013, pp. 693-702.

[6] F B. Sazoglu, B B. Cambazoglu, R. Ozcan, et al. “Strategies for setting time-to-live values in result caches”. The 22nd ACM International Conference on Information and Knowledge Management, SanFrancisco, CA, USA, 2013, pp. 1881-1884.

[7] R. Blanco, E. Bortnikov, F. Junqueira, et al. “Caching Search Engine Results over Incremental Indices”. The 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, Geneva, Switzerland, 2010, pp.82-89.

[8] J. Zhang, X. Long, and T. Suel, “Performance of compressed inverted list caching in search engines”, The 17th International Conference on World Wide Web, Beijing, China, 2008, pp. 387-396.

[9] R. Baeza-Yates, A. Gionis, and F. Junqueira, “The impact of caching on search engines”, The 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, Netherlands, 2007, pp. 183-190.

[10] X. Long, T. Suel, “Three-level caching for efficient query processing in large web search engines”, The 14th International Conference on World Wide Web, Chiba, Japan, 2005, pp. 369-395.

[11] R. Ozcan, I S. Altingovde, and B B. Cambazoglu, “A five-level static cache architecture for web search engines”, Information Processing and Management, 5th ed, vol 48, 2012, pp. 828-840.

[12] E. Feuerstein and G. Tolosa. “Cost-aware intersection caching and processing strategies for in-memory inverted indexes”. In Proc. of 11th Workshop on Large-scale and Distributed Systems for Information Retrieval. New York, 2014, pp. 413-440.

[13] G. Tolosa, L. Becchetti, E. Feuerstein, et al. “Performance Improvements for Search Systems Using an Integrated Cache of Lists+Intersections”. The 21st International Symposium on String Processing and Information Retrieval, Ouro Preto, Brazil, 2014, pp. 227-235.

[14] J. Wang, E. Lo, M L. Yiu. et al. “Cache Design of SSD-based Search Engine Architectures: An Experimental Study”. ACM Transactions on Information System, 4th ed, vol 32, 2014, pp. 1-26.

[15] D. Ceccarelli, C. Lucchese, and S. Orlando, “Caching query-biased snippets for efficient retrieval”, 14th International Conference on Extending Database Technology, Uppsala, Sweden, 2011, pp. 93-104.

[16] A. Turpin, Y. Tsegay, and D. Hawking, “Fast generation of result snippets in web search”, Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, 2007, pp. 127-134.

[17] R. AgrawaI, R. Srikant, “Fast algorithms for mining association for mining association rules”, The 20th international Conference on very large database, Santiage de Chile, Chile, 1994, pp. 487 - 499.

[18] J. Han, J. Pei, and Y. Yin, “Mining Frequent Patterns without Candidate Generation”, The 2000 ACM SIGMOD International Conference on Management of Data, Dallas, Texas, 2000, pp. 1 - 12.

[19] Y. Cheung, A W. Fu, “Mining Frequent Itemsets without Support Threshold: With and without Item Constraints”, IEEE Transactions on Knowledge and Data Engineering, 6th ed, vol 16, 2004, pp. 1052-1069.

[20] J. Lee, C W. Clifton. “Top-k frequent itemsets via differentially private FP-trees”. The 20th ACM SIGKDD international conference on Knowledge discovery and data mining, New York, NY, USA, 2014, pp. 931-940.