Using Big Data Analysis to Improve Cache Performance in Search Engines

Gabriel Tolosa[1,2] and Esteban Feuerstein[1]

[1] University of Buenos Aires, Argentina
[2] National University of Luján, Argentina
tolosoft@unlu.edu.ar, efeuerst@dc.uba.ar

Abstract. Web Search Engines process huge amounts of data to support search but must run under strong performance requirements (to answer a query in a fraction of a second). To meet that performance they implement different optimization techniques such as caching, that may be implemented at several levels. One of these caching levels is the *intersection cache*, that attempts to exploit frequently occurring pairs of terms by keeping in the memory of the search node the results of intersecting the corresponding inverted lists. In this work we propose an optimization step to decide which items should be cached and which not by introducing the usage of data mining techniques. Our preliminary results show that it is possible to achieve extra cost savings in this already hyper-optimized field.

Keywords: big data, search engines, intersection caching, classification

1 Introduction

Web Search Engines (WSE) are probably one of the first examples of Big Data technology. From their beginnings, they needed to process huge amounts of data (i.e. web pages) to build sophisticated structures that support search. The (big) storage, analysis, management and search challenges that they proposed lead to the development of distributed architectures and algorithms that scale up to many billion documents. Many crawling, mining, linking and search algorithms form the basis of modern search technologies.

To handle the increasing scale and variety of data, WSE architectures are organized in clusters of commodity hardware that offer the possibility to easily add/replace machines. This kind of cluster is formed by a front-end node (*broker*) and a large number of *search nodes* that process queries in parallel. According to [3] a web-scale search engine cluster is normally made up of more than 15.000 commodity class PCs.

Each search node holds only a fraction of the document collection and builds local data structures (inverted indices) that are used to achieve high query throughput. Given a cluster of P search nodes and a document collection of C documents, each node maintains an index with information related to only $\frac{C}{P}$ documents (assuming an even document distribution among nodes). Besides, to

meet the strong performance requirements typically imposed to WSE (shortly: to answer a query in a fraction of a second), WSEs usually implement different optimization techniques where one of the most important is caching [2].

Caching can be implemented at several levels in a typical achitecture. At broker level, a *results cache* [7] is generally maintained. This stores the final list of results corresponding to the most frequent or recent queries. The lowest level is *posting list caching* which simply stores in a memory buffer the posting lists of popular or valuable terms. Complementarily, an *intersection cache* [6] may be implemented to obtain additional performance gains. This cache attempts to exploit frequently occurring pairs of terms by keeping in the memory of the search node the results of intersecting the corresponding inverted lists (saving not only disk access time but CPU time as well).

In previous work we showed that it is possible to obtain extra savings increasing the efficiency and effectiveness [5] of the intersection cache and proposed hybrid data structures [8]. In this work we go a step ahead in that direction introducing the usage of data mining techniques to decide which items should be cached and which not.

2 Proposal and Preliminary Results

Given a conjunctive query $q = \{t_1, t_2, t_3, ..., t_n\}$ that represents a user's information need (each t_i is a term, and the user wants the list of documents that contain *all* the terms) we adopt a query resolution strategy that first decomposes the query in pairs of terms. Each pair is checked in the Intersection Cache and the final resolution order is given by first considering the pairs that are present in the cache and afterwards intersecting them with the remaining ones [5]. The intersection cache is dynamically managed using the Greedy-Dual Size (GDS) strategy [4], and an interesting challenge is to define an access policy, i.e. to previously decide which pairs will be allowed into the cache and which ones will not. In this work we take up that challenge.

We model this as a classification problem with the added restriction that we must use the minimal number of features because this decision should not affect the overall performance of the search node, even at the expense of lower classification ratios. Our first attempt is to identify pairs of terms that appear only once in the query stream, and therefore caching them does not provide any advantage. We call these *singleton* pairs. We consider only four features of each pair (t_1, t_2):

- TF_{t1} and TF_{t2}: The individual term frequencies of t_1 and t_2 (respectively) in a query log used as a training set.
- DF_{t1} and DF_{t2}: The document frequencies of t_1 and t_2 in the document collection that the search node holds.

For our experiments we use a document collection derived from the Stanford WebBase Project of roughly 8 million documents (appropriate for a single node) and the well-known AOL query log. We train our classifiers with 40 million pairs

of terms of positive and negative examples and tested several approaches. We report here the results of two of them: Decision Trees (DT) and Random Forest (RD) [1].

Given a trained classifier, a pair of terms (t_1, t_2) and the values of the observed features of the pair, the decision process is quite simple:

```
if not (isSingleton(pair, TF_t1, TF_t2, DF_t1, DF_t2))
   insert-in-cache (pair)
else
   do nothing
```

Our baseline is the basic caching algorithm without any cache access policy (NoAP) and the ideal bound is a variation of a *clairvoyant* algorithm that "knows" all the singleton pairs and avoids caching them (AP-Clair). We explore several cache sizes and run 5 million queries over a simulation framework [5]. Figure 1 shows the results we obtain.

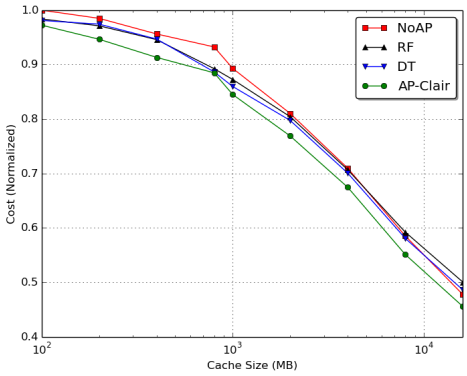

Fig. 1. Performance of the different policies (x-axis in log scale).

Although the classification performance was rather low (75% in the best case) our results show that with this simple data mining approach to handle the access to the Intersection Cache it is possible to achieve extra cost savings in this already hyper-optimized field. The access policies achieve an improvement of around 3% for cache sizes smaller than 8 GB (AP-Clair reduces the cost by 6% in the best case). The Decision Tree-based classifier performed slightly better than the Random Forest method. We think this happens because, in our case, a reduced number of features is considered. The resulting tree does not grow enough to overfit the training set thus the advantages of the forest approach are not significant. Presently, we are using this feedback in our algorithms to improve the classification performance and adding new strategies to detect singleton pairs more accurately.

References

1. C. Aggarwal. *Data Classification: Algorithms and Applications.* Chapman and Hall/CRC Press, 2014.
2. R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. In *Proc. of the 30th annual Int. Conf. on Research and Development in Information Retrieval*, SIGIR '07, pages 183–190, USA, 2007.
3. L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, Mar. 2003.
4. P. Cao and S. Irani. Cost-aware www proxy caching algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, USITS'97, pages 18–18, Berkeley, CA, USA, 1997. USENIX Association.
5. E. Feuerstein and G. Tolosa. Cost-aware intersection caching and processing strategies for in-memory inverted indexes. In *Workshop on Large-scale and Distributed Systems for Information Retrieval (LSDS-IR 2014) at ACM International Conference on Web Search and Data Mining (WSDM'14)*, LSDS-IR'14, New York, NY, USA, 2014. ACM.
6. X. Long and T. Suel. Three-level caching for efficient query processing in large web search engines. In *Proc. of the 14th Int. Conf. on World Wide Web*, WWW '05, pages 257–266, USA, 2005.
7. E. Markatos. On caching search engine query results. *Comput. Commun.*, 24(2):137–143, Feb. 2001.
8. G. Tolosa, L. Becchetti, E. Feuerstein, and A. Marchetti-Spaccamela. Performance improvements for search systems using an integrated cache of lists+intersections. In *Proceedings of 21st International Symposium of String Processing and Information Retrieval, SPIRE'14*, pages 227–235, 2014.