Modeling Static Caching in Web Search Engines*

Ricardo Baeza-Yates[1] and Simon Jonassen[2]

[1] Yahoo! Research Barcelona
Barcelona, Spain
[2] Norwegian University of Science and Technology
Trondheim, Norway

Abstract. In this paper we model a two-level cache of a Web search engine, such that given memory resources, we find the optimal split fraction to allocate for each cache, results and index. The final result is very simple and implies to compute just five parameters that depend on the input data and the performance of the search engine. The model is validated through extensive experimental results and is motivated on capacity planning and the overall optimization of the search architecture.

1 Introduction

Web search engines are crucial to find information among more than 180 million Web sites active at the end of 2011[3], and users expect to rapidly find good information. In addition, the searchable Web becomes larger and larger, with more than 50 billion static pages to index, and evaluating queries requires the processing of larger and larger amounts of data each day. In such a setting, to achieve a fast response time and to increase the query throughput, using a specialized cache in main memory is crucial.

The primary use of a cache memory is to speedup computation by exploiting frequently or recently used data. A secondary but also important use of a cache is to hold pre-computed answers. Caching can be applied at different levels with increasing response latencies or processing requirements. For example, the different levels may correspond to the main memory, the disk, or resources in a local or a wide area network. In the Web, caching can be at the client side, the server side, or in intermediate locations such as a Web proxy [14].

The cache can be static or dynamic. A static cache is based on historical information and can be periodically updated off-line. If the item that we are looking for is found in the cache, we say that we have a hit, otherwise we say that we have a miss. On the other hand, a dynamic cache replaces entries according to the sequence of requests that it receives. When a new request arrives, the cache system has to decide whether to evict some entry from the cache in the

* This work was done while the second author was an intern at Yahoo! Research and supported by the iAd Centre (http://iad-centre.no) funded by the Research Council of Norway and the Norwegian University of Science and Technology.

[3] According to Netcraft, January 2012.

case of a cache miss. These decisions are based on a cache policy, and several different policies have been studied in the past.

In a search engine there are two possible ways to use a cache memory:

Caching results: As the engine returns results to a particular query, it may decide to store these results to resolve future queries. This cache needs to be periodically refreshed.

Caching index term lists: As the search engine evaluates a particular query, it may decide to store in memory the inverted lists of the involved query terms. As usually the whole index does not fit in memory, the engine has to select a subset to keep in memory and speed up the processing of queries.

For designing an efficient caching architecture for web search engines there are many trade-offs to consider. For instance, returning an answer to a query already existing in the cache is much more efficient than computing the answer using cached inverted lists. On the other hand, previously unseen queries occur more often than previously unseen terms, implying a higher miss rate for cached results. Caching of inverted lists has additional challenges. As inverted lists have variable size, caching them dynamically is not very efficient, due to the complexity involved (both in efficiency and use of space) and the skewed distribution of the query stream. Neither is static caching of inverted lists a trivial task: when deciding which terms to cache one faces the trade-off between frequently queried terms and terms with small inverted lists that are space efficient. Here we use the algorithm proposed by Baeza-Yates *et al.* in [2]. In that paper it is also shown that in spite that the query distribution changes and there are query bursts, the overall distribution changes so little that the static cache can be precomputed every day without problems. This paper also leaves open the problem on how to model the optimal split of the cache between results and inverted lists of the index, which we tackle here.

In fact, in this paper we model the design of the two cache level explained before, showing that the optimal way to split a static cache depends in a few parameters coming from the query and text distribution as well as on the exact search architecture (e.g. centralized or distributed). We validate our model experimentally, showing that a simple function predicts a good splitting point. In spite that cache memory might not be expensive, using this resource well does change the Web search engine efficiency. Hence, this result is one component in a complete performance model of a Web search engine, to do capacity planning and fine tuning of a given Web search architecture in an industrial setting.

The remainder of this paper is organized as follows. Section 2 covers related work while Section 3 shows the characteristics of the data that we used to find the model as well as perform the experimental validation. Section 4 presents our analytical model while Section 5 presents the experimental results. We end with some conclusions in Section 6.

2 Related Work

Query logs constitute a valuable source of information for evaluating the effectiveness of caching systems. As first noted by Xie and O'Hallaron [18], many popular queries are shared by different users. This level of sharing justifies the choice of a server-side caching system in Web search engines. One of the first papers on exploiting user query history was proposed by Raghavan and Sever [15]. Although their technique is not properly caching, they suggest using a *query base*, built upon a set of persistent "optimal" queries submitted in the past, in order to improve the retrieval effectiveness for similar future queries. That is, this is a kind of static result cache. Later, Markatos [10] shows the existence of temporal locality in queries, and compares the performance of different caching policies.

Based on the observations of Markatos, Lempel and Moran proposed an improved caching policy, Probabilistic Driven Caching, based on the estimation of the probability distribution of all possible queries submitted to a search engine [8]. Fagni *et al.* follow Markatos' work by showing that combining static and dynamic caching policies together with an adaptive prefetching policy achieves a high hit ratio [6].

As search engines are hierarchical system, some researchers have explored multi-level architectures. Saraiva *et al.* [16] proposes a new architecture for web search engines using a dynamic caching system with two levels, targeted to improve response time. Their architecture use an LRU policy for eviction in both levels. They find that the second-level cache can effectively reduce disk traffic, thus increasing the overall throughput. Baeza-Yates and Saint-Jean propose a three level index organization for Web search engines [5], similar to the one used in current architectures. Long and Suel propose a caching system structured according to three different levels [9]. The intermediate level contains frequently occurring pairs of terms and stores the intersections of the corresponding inverted lists. Skobeltsyn *at al.* [17] adds a pruned index after the result cache showing that this idea is not effective as inverted list caching basically serves for the same purpose.

Later, Baeza-Yates *et al.* [2, 3] explored the impact of different static and dynamic techniques for inverted list caching, introducing the QtfDf algorithm for static inverted list caching that we use here. This algorithm improves upon previous results on dynamic caching with similar ideas [9].

More recent work on search engine caching includes how to avoid cache pollution in the dynamic cache [4] and how to combine static and dynamic caching [1]. Gan and Suel improve static result caching to optimize the overall processing cost and not only the hit ratio [7], an idea also explored by Ozcan *et al.* [13] for dynamic caching. In a companion paper, Ozcan *et al.* [12] introduce a 5-level static caching architecture to improve the search engine performance.

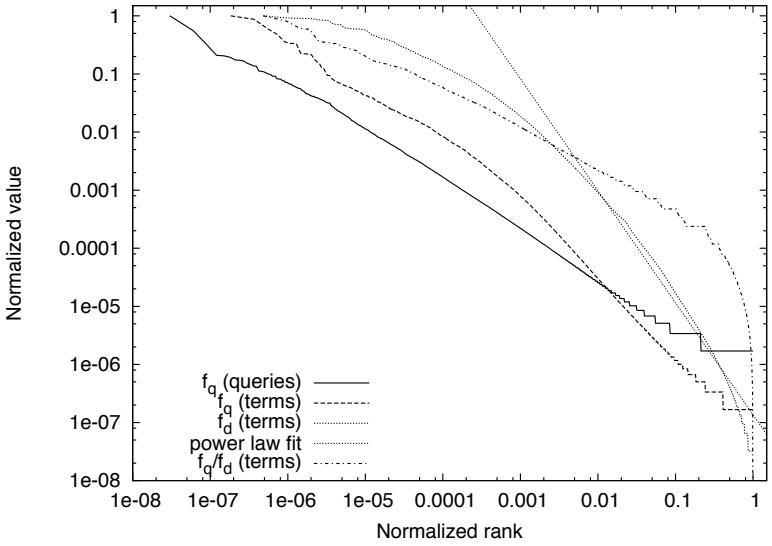

Fig. 1. Queries and terms distribution as well as the query-text term frequency ratio.

3 Data Characteristics

For the experiments we use a 300GB crawl of the UK domain from 2006, which contains 37.9 million documents, and a random sample of 80.7 million queries submitted to a large search engine between April 2006 and April 2007. In both the query log and the document collection, we normalize tokens, replace punctuation characters with spaces and remove special characters. The complete query log contains 5.46 million unique terms, with 2.07 million of which appear in the document collection. For these terms we count the total number of documents (document frequency) where each of these terms appears. Finally, we split the query log into two logs - a training log containing the first 40 million queries and a testing log for evaluation containing the remaining 40.3 million queries.

The query distribution as well as the term distribution in the queries and in the Web collection are shown in Figure 1. In this figure we also show the distribution of the ratio of the query and text term frequencies, as this is the heuristic used in the static inverted list caching algorithm (QtfDf), which fills the cache using the inverted lists in decreasing order of this ratio. All these distributions follow a power law in their central part and hence a good approximation for those curves is k/r^{γ}, where r is the item rank in decreasing order. Table 1 shows these parameters for the different variables, where u is the overall fraction of unique items, a value needed later.

Table 1. Characteristics of the power law distributions.

Variable	γ	k	u
Queries	0.8635	$1.1276 \cdot 10^{-6}$	0.466
Query terms	1.3532	$1.0388 \cdot 10^{-3}$	0.039
Text terms	1.9276	$1.3614 \cdot 10^{-7}$	
Tfq-Tfd ratio	0.7054	$2.0328 \cdot 10^{-4}$	

Notice that the γ value for the distribution of the ratio of term frequencies is almost the ratio of the γ values of the two distributions involved in spite that the correlation between both term distributions is only 0.4649.

4 Modeling the Cache

Our static cache is very simple. We use part of the cache for results and the rest for the inverted lists of the index. Our memory cache do not need to be in the same server. That is, we have an overall amount of memory, that can be split into two parts. Usually the results cache will be closer to the client (e.g. in the front end Web server or a proxy) and the index cache could in a local (centralized case) or a remote (distributed or WAN case) search server. Therefore, our problem is how to split the memory resources in the best possible way to minimize the search time.

We first need to model the hit ratio, that is, the probability of finding a result in the cache for a query. As shown in the previous section, the query distribution follows a power law. Hence, the hit ratio curve will follow the surface under the power law. That is, if we approximate the query distribution by the function k/r^{γ}, the hit ratio function will follow a function of the form $k'r^{1-\gamma}$ reaching a limit for large r as all unique queries will always be missed (in most query logs unique queries will be roughly 50% of the volume). This hit ratio curve is shown for the result cache in Figure 2 (top) and the limit is $1 - u$ from Table 1.

The optimal split for a given query distribution is not trivial as we have two cascading caches and the behavior of the second will depend on the performance of the first. Hence, modeling this dynamic process is quite difficult. However, based in our previous work [17] we notice that the query distribution after the result cache had basically the same shape as the input query distribution. We corroborate that finding, plotting in Figure 2 (bottom) the query distribution after the result cache for different cache sizes. So an approximate model is to assume that the hit ratio curve for inverted lists is independent of the result cache. As the distribution of cached inverted lists also follows a power law, we use the same function as the hit ratio curve for the result cache, and only the parameters change. That is, the hit ratio in both cases is modeled by

$$h(x) = \frac{k}{x^{\gamma}}$$

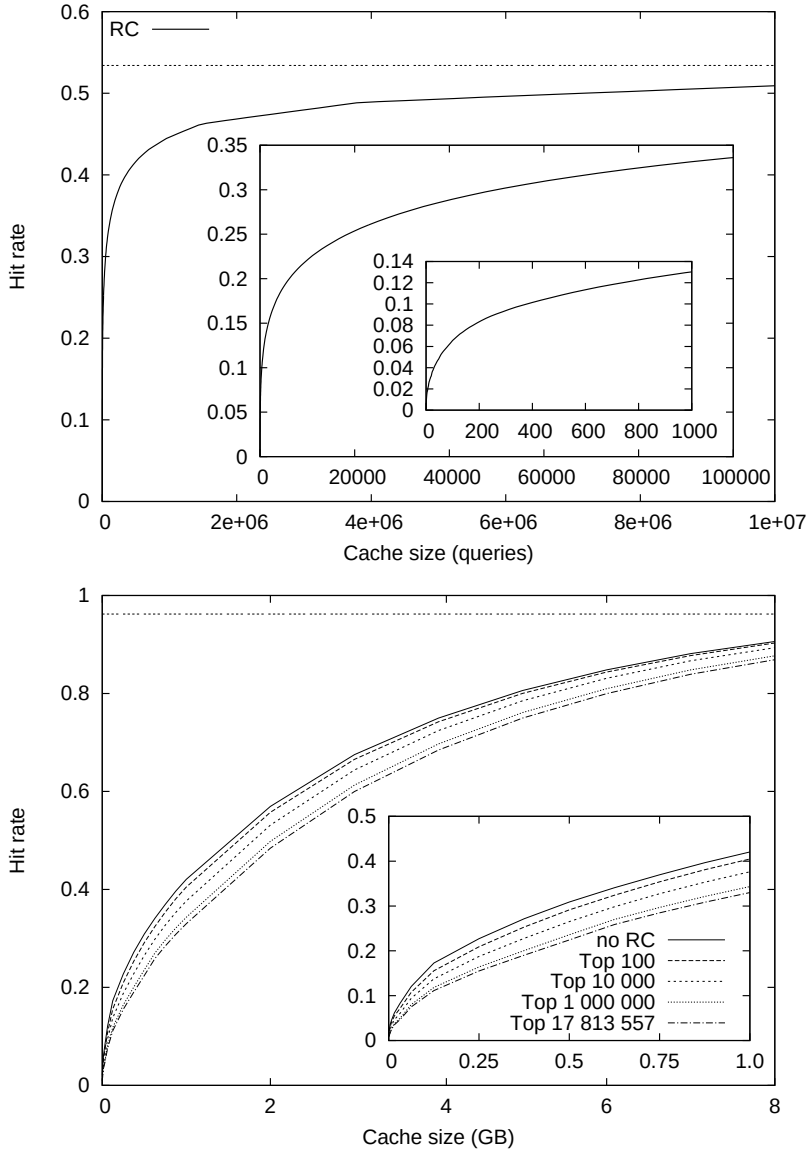

Fig. 2. Hit ratio in the results (top) and the inverted list (bottom) cache.

where x is the cache fraction used. For large x, $h(x)$ converges to the limit $1-u$, and we force that by defining $h(1) = 1 - u$ which then sets the value of k. As we will see later, this second approximation does not affect much the analytical results and is similar to the infinite cache case. Notice that we have already found the experimental values of u and γ in Section 3.

Assuming that this model is a good approximation, we can use it to find the optimal split fraction for this case. Let x be that optimal point and h_R and h_I the corresponding hit functions. Hence, the search time is given by

$$T(x) = t_R\ h_R(x)\ +\ t_I\ (h_I(1-x) - h_R(x))\ +\ t_P\ (1 - h_I(1-x))$$

where t_R, t_I, and t_P are the average response time of the search engine when the query is answered from the result cache, the index cache or has to be fully processed. Notice that for the range of interest (e.g. $x > 0.5$) and that given that in practice h_I is much larger than h_R as we will see in the next section, the second term is always positive.

Simplifying the previous formula and computing the partial derivative of the resultant expression with respect to x, we obtain that the optimal split must satisfy the following equation

$$\frac{(1-x)^{\gamma_L}}{x^{\gamma_R}} = \frac{k_L\ (t_P - t_I)(1-\gamma_L)}{k_R\ (t_I - t_R)(1-\gamma_R)}$$

which can be solved numerically. However, by expanding the left hand side, and considering the relative answer time (that is, we set $t_R = 1$) and that $t_I >> t_R$ in the right hand side, we obtain that the split point x^* can be approximated by a simple function

$$x^* = \frac{1}{TRr^{1+\gamma_L/\gamma_R}}\ ,$$

where $TRr = t_P/t_I$. In the following section we validate this model.

5 Experimental Validation

We performed experiments simulating both cases, the centralized and the distributed case. In the centralized case the front end Web server is directly connected to the search server. The distributed architecture implies a local front end connected to a remote server through an Internet connection. We did not consider the local area network case (e.g. the case of a cluster) because the results were basically the same as in the centralized case. For comparison purposes we used the processing times of [2], shown in Table 2. Here we consider only the compressed case as this is what it is used in practice, but we compare our model also with the previous uncompressed results in the next section. We also consider two cases when processing the query: full evaluation (compute all possible answers) and partial evaluation (compute the top-10k answers). The two-level cache was implemented using a fraction of the main memory available with the result cache in the front end Web server and the index cache in the search server. We do not consider the refreshing of the result cache as there are orthogonal techniques to do it, and that refreshing the stored results do not change the queries stored in the cache.

We model the size of compressed inverted lists as a function of the document frequency. In our previous work [2] we used the Terrier system [11] that uses

Table 2. Ratios between the average time to evaluate a query and the average time to return cached results for the centralized and distributed cases with full or partial evaluation, and uncompressed (1Gb cache) and compressed (0.5Gb cache) index.

System	Uncompressed		Compressed	
Centralized	TR_1^C	TR_2^C	$TR_1'^C$	$TR_2'^C$
Full evaluation	233	1760	707	1140
Partial evaluation	99	1626	493	798
Distributed	TR_1^D	TR_2^D	$TR_1'^D$	$TR_2'^D$
Full evaluation	5001	6528	5475	5908
Partial evaluation	4867	6394	5270	5575

gamma encoding for document ID gaps and unary codes for frequencies. Here we use the state of the art, compressing the inverted lists with the NewPFor approach by Zhang, Long and Suel [19]. We have analyzed both techniques in the TREC GOV2 corpus and NewPFor is in general a bit better. In this method inverted lists are grouped into blocks of 128 entries and compressed together - one block of compressed d-gaps is followed by a block of compressed frequencies. Blocks shorter than 100 entries are compressed using VByte. In the following we use NewPFor compression which we approximate with the following function:

$$sizeB(docs) = \begin{cases} \text{if } docs < 100 \text{ then} & 1.81 + 3.697 \cdot docs \\ \text{else} & 176.42 + 2.063 \cdot docs \end{cases}$$

For the results cache we use entries of size 1264 bytes.

We consider cache memories that are a power of 2 starting with 128Mb and finishing in 8Gb, using query frequency in decreasing order to setup the results cache and using the $QtfDf$ algorithm to setup the index cache. The setup is done with the training query log while the experimental results are done with the testing query log that were described in Section 3. In Figure 3 we show one example for the response time for a memory cache of 1Gb in function of the cache size used for results. In the centralized case the optimal slit is in 0.62 while in the distributed case is almost 1. In Figure 4 we show the optimal fraction for the results cache in function of the overall cache size. As the results cache size reaches a saturation point, the optimal fraction decreases while the overall cache size increases.

We also tried a variant of the static caching algorithm by not considering terms with $f_q < A$ and then using a modified weight: $f_q / \min(f_d, B)$ with different A and B in the set $\{100, 1000, 10000\}$. This did improve the performance but only for queries involving very long inverted lists.

In Figure 5 we compare our experimental results and the results of our previous work [2] with our model and our approximated solution depending on TRr which is the ratio $t_P/t_I > 1$ for the different cases. The predicted optimal ratio is quite good, in particular for partial evaluation (lower data points) which is also the most realistic case in a Web search engine.

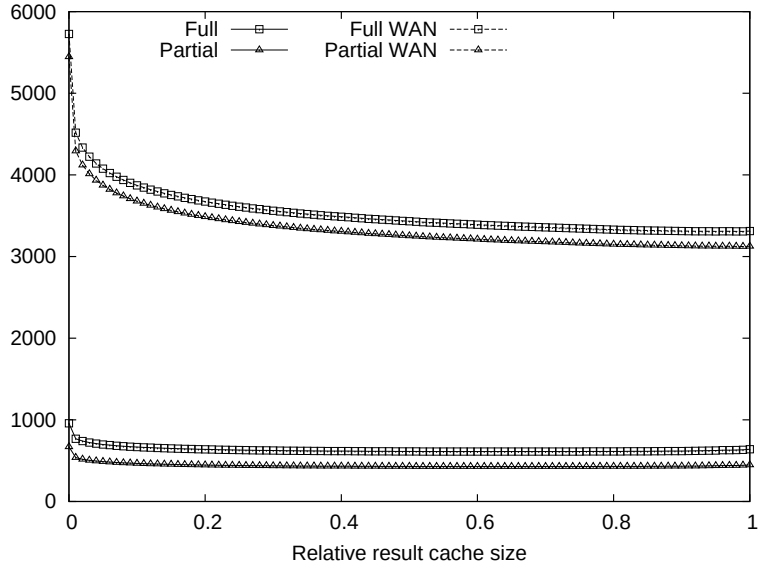

Fig. 3. Answer time performance for different splits in a cache of 1GB.

6 Conclusions

We have shown that to split almost optimally a static cache, for our Web search engine, we just need to compute five parameters: the power law exponent estimation for the distribution of queries and query terms as well as for document terms, plus the average response time when answering with the index cache or when processing the whole query. Notice that this assumes that we can compute the power law of the query-document term ratio with just a division. Otherwise a sixth parameter needs to be computed.

Further work includes doing further experimental results with more query log samples, using those results to improve this model and later extend it to more complex static caching schemes [7, 12].

References

1. Altingovde, I., Ozcan, R., Cambazoglu, B., Ulusoy, O.: Second chance: A hybrid approach for dynamic result caching in search engines. In: Clough, P., Foley, C., Gurrin, C., Jones, G., Kraaij, W., Lee, H., Mudoch, V. (eds.) ECIR 2011, Lecture Notes in Computer Science, vol. 6611, pp. 510–516. Springer Berlin / Heidelberg (2011)
2. Baeza-Yates, R.A., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: The impact of caching on search engines. In: SIGIR. pp. 183–190 (2007)
3. Baeza-Yates, R.A., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: Design trade-offs for search engine caching. TWEB 2(4) (2008)

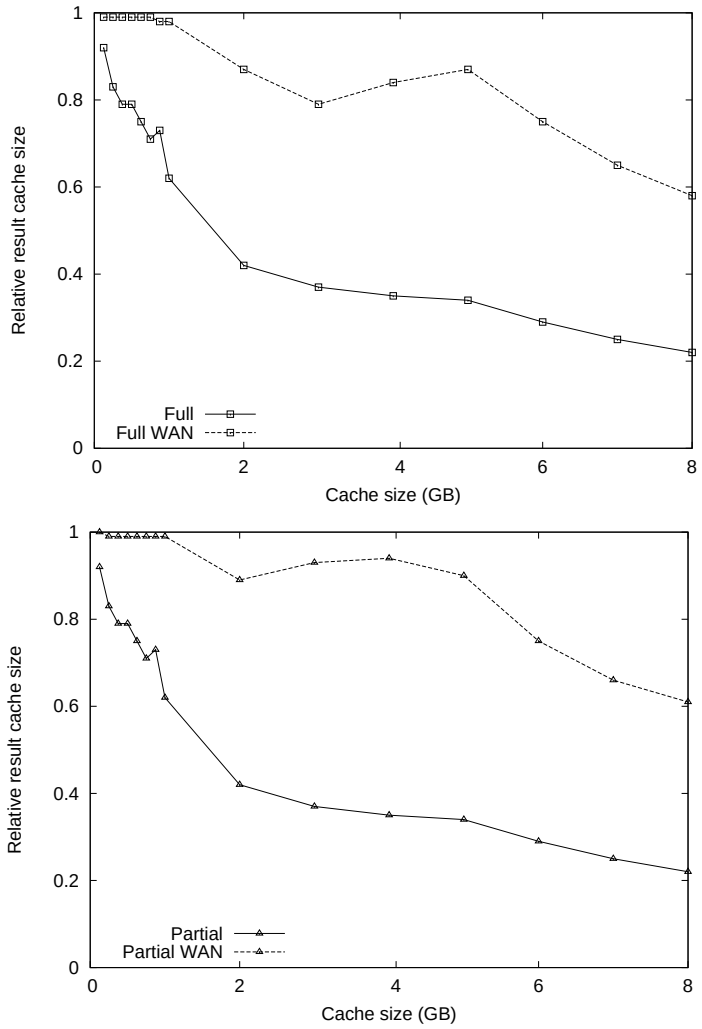

Fig. 4. Optimal split depending on the cache size for full (top) and partial (bottom) evaluation.

4. Baeza-Yates, R.A., Junqueira, F., Plachouras, V., Witschel, H.F.: Admission policies for caches of search engine results. In: SPIRE. pp. 74–85 (2007)
5. Baeza-Yates, R.A., Saint-Jean, F.: A three level search engine index based in query log distribution. In: SPIRE. pp. 56–65 (2003)
6. Fagni, T., Perego, R., Silvestri, F., Orlando, S.: Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. ACM Trans. Inf. Syst. 24(1), 51–78 (2006)
7. Gan, Q., Suel, T.: Improved techniques for result caching in web search engines. In: WWW. pp. 431–440 (2009)

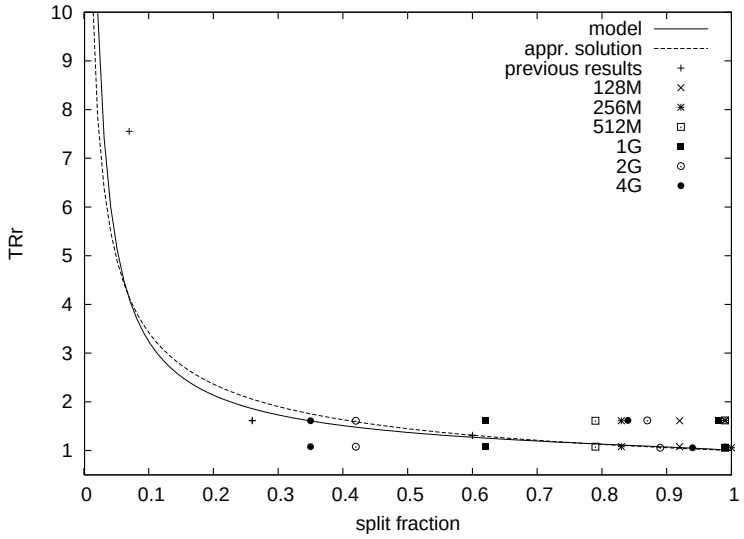

Fig. 5. Comparison of the experimental results and the model.

8. Lempel, R., Moran, S.: Predictive caching and prefetching of query results in search engines. In: WWW (2003)
9. Long, X., Suel, T.: Three-level caching for efficient query processing in large web search engines. In: WWW (2005)
10. Markatos, E.P.: On caching search engine query results. Computer Communications 24(2), 137–143 (2001), citeseer.ist.psu.edu/markatos00caching.html
11. Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., Johnson, D.: Terrier information retrieval platform. In: ECIR. pp. 517–519 (2005)
12. Ozcan, R., Altingovde, I.S., Cambazoglu, B.B., Junqueira, F.P., Özgür Ulusoy: A five-level static cache architecture for web search engines. Information Processing & Management (2011), http://www.sciencedirect.com/science/article/pii/S0306457310001081, in press
13. Ozcan, R., Altingovde, I.S., Ulusoy, O.: Cost-aware strategies for query result caching in web search engines. ACM Trans. Web 5, 9:1–9:25 (May 2011)
14. Podlipnig, S., Boszormenyi, L.: A survey of web cache replacement strategies. ACM Comput. Surv. 35(4), 374–398 (2003)
15. Raghavan, V.V., Sever, H.: On the reuse of past optimal queries. In: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval. pp. 344–350 (1995)
16. Saraiva, P.C., de Moura, E.S., Ziviani, N., Meira, W., Fonseca, R., Riberio-Neto, B.: Rank-preserving two-level caching for scalable search engines. In: SIGIR (2001)
17. Skobeltsyn, G., Junqueira, F., Plachouras, V., Baeza-Yates, R.A.: Resin: a combination of results caching and index pruning for high-performance web search engines. In: SIGIR. pp. 131–138 (2008)
18. Xie, Y., O'Hallaron, D.R.: Locality in search engine queries and its implications for caching. In: INFOCOM (2002)
19. Yan, H., Ding, S., Suel, T.: Inverted index compression and query processing with optimized document ordering. In: WWW. pp. 401–410 (2009)