Inverted List Caching for Topical Index Shards

Zhuyun Dai and Jamie Callan

Language Technologies Institute, Carnegie Mellon University
{zhuyund, callan}@cs.cmu.edu

Abstract. Selective search is a distributed retrieval architecture that intentionally creates skewed postings and access patterns. This work shows that the well-known `QtfDf` inverted list caching algorithm is as effective with topically-partitioned indexes as it is with randomly-partitioned indexes. It also shows that a mixed global-local strategy reduces total I/O without harming query hit rates.

1 Introduction

Search engines use caching to reduce computation and disk access. One form of caching, *list caching*, keeps the inverted lists of frequent query terms in memory. Usually it is used by distributed search architectures that *randomly* partition the corpus into index shards, maintain one list cache per shard, and assign one shard per processor. Thus postings are distributed uniformly across index shards, list caches, and processors.

Selective search is a distributed search architecture that partitions a corpus into many *topic-based* index shards and routes each query to the few shards that are most likely to have relevant documents [1, 4, 5]. Topic-based indexes skew postings distributions and access patterns. Few index shards are searched per query, so it is not necessary to have a one-to-one mapping between shards and processors. Instead, multiple topic-based index shards are assigned to each processor. These differences create a substantially different caching environment - one in which data and access patterns are more skewed, and multiple index shard caches compete for a processor's RAM.

It is an open question whether inverted list caching is effective with topic-based index shards. This paper investigates the behavior of the well-known `QtfDf` static caching algorithm [2] when used with selective search. First it investigates whether topically indexed shards reduce the impact of `QtfDf` caching. Second, it investigates whether using shard-wise query log information can improve list caching for selective search.

2 Related Work

Selective search partitions a collection into topic-based index shards that have skewed vocabularies and access patterns. Resource selection algorithms (e.g., Rank-S [5], Taily [1]) select shards to search for each query. Shards are searched in parallel; results are merged to form a final ranking. Computational costs are lower than with traditional distributed retrieval because fewer and smaller shards are searched for any query [4, 5].

Inverted list caching has been studied extensively [2, 6, 7, 8]. Most studies considered a centralized system or a traditional distributed system in which queries are run on all shards. `QtfDf` [2] is a well-known static caching policy for posting lists. It caches

the posting lists of the terms with the highest values of the ratio $\frac{qtf}{df}$, where qtf denotes the frequency of the term in a query log, and df the document frequency of the term. Experiments showed that although `QtfDf` is a static policy, it can outperform dynamic policies because the query term vocabulary is stable over time [2].

3 Caching Strategies

A distributed system has a total of C CPU cores. Each core has a cache of size m. With random document partitioning, there are C shards, and each core serves one shard. With topical partitioning, there are P shards, typically with $P \gg C$, and each core serves multiple shards. The cache is a key-value data structure. The key is (shard s, term t), and the corresponding value is the posting list of term t in shard s.

`QtfDf` [2] uses term frequency in a query log (qtf) and document frequency in the corpus (df) to select terms for the cache. It is a *global* strategy because it does not use shard-specific statistics. We refer to it as `QtfDf-G` to distinguish it from local methods.

$$\texttt{QtfDF-G}(t) = \frac{qtf(t)}{df(t)} \tag{1}$$

Each CPU core caches the posting lists of the highest scoring terms from its shard(s). When a CPU core hosts multiple shards, they share its cache. Each *topical* shard uses a different amount of cache because a term's posting list lengths vary across shards. We found it to be more effective than forcing topical shards to use equal amounts of cache. Thus, topical and random partitions do not cache identically under `QtfDf-G`.

When shards are partitioned randomly, global qtf and df are representative for individual shards. However, topical shards have skewed contents and access patterns. A term may not be equally valuable to each shard. The second strategy estimates the value of term t to shard s.

$$\texttt{QtfDf-L}(t, s) = \frac{qtf_s(t)}{df_s(t)} \tag{2}$$

$qtf_s(t)$ is the term frequency of term t in queries submitted to shard s. $df_s(t)$ is the number of documents in shard s that contain term t. Each core ranks its (shard s, term t) pairs, and loads the highest scoring posting lists into the cache until it is full.

4 Experimental Methodology

Document collection: We used ClueWeb09-B which has 50M documents, 96M unique terms, and 109GB of uncompressed postings. Each posting is a pair of (docid, tf).

Selective Search settings followed prior research. The collection was distributed on two 8-core machines (16 cores in total). For exhaustive search, each CPU core served one index partition; documents were distributed *randomly* across 16 partitions. For selective search, the document collection was partitioned into 123 *topical* index shards using `QKLD-QInit`[1] [3]. Shards were assigned to the 16 cores by random and log-based

[1] Obtained from http://boston.lti.cs.cmu.edu/appendices/CIKM16-Dai/CW09B-cent1/

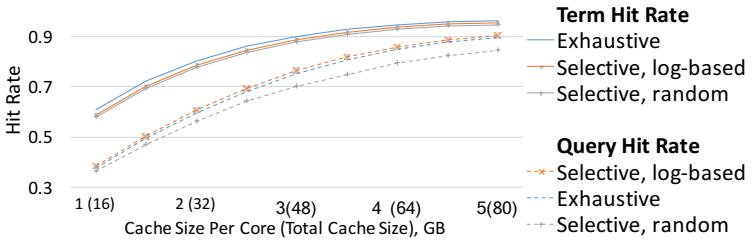

Fig. 1: Term hit rate (solid lines) and query hit rate (dashed lines) of `QtfDF-G` under different shard assignment policies. Average number of shards searched per query: $n = 3.9$

policies [4]. Taily [1] was used for resource selection with the default parameters. It selected 3.9 shards per query on average. For higher recall rate, we also tested Taily with a lower threshold, which selected an average of 8.5 shards per query.

Query log: We used the AOL query log. Queries from the first 3 weeks were used to populate caches; queries from the next 3 weeks were used for testing. Queries that appeared less than 10 times were filtered out. The training queries had 0.3 million unique terms covering 86GB of posting lists.

Evaluation: Two metrics were used. **Term hit rate** is the percentage of times a requested term was in the cache for a selected shard. **Query hit rate** is the percentage of queries that had *all* terms in cache for *all* selected shards. Term hit rate measures the effect on posting list I/O. Query hit rate measures the effect on response times. It is harder to achieve high query hit rate than high term hit rate.

5 Experimental Results

5.1 Global Term Selection (`QtfDf-G`)

First we tested the global method `QtfDf-G` on random (*exhaustive*) and topical (*selective*) shards. There is a many-to-one mapping of topical shards to cores, thus selective search requires a shard assignment policy. Most prior research randomly assigned shards to cores [5], but log-based policy improves throughput by assigning shards by their popularity [4]; we tested both.

`QtfDf-G` is equally effective for exhaustive search and selective search with *log-based* assignment (Fig. 1). `QtfDf-G` is less effective for selective search with *random* shard assignment, especially for query hit rate. Similar trends were observed in the high-recall setting that searched an average of $n = 8.5$ shards per query (not shown).

Random assignment had a lower hit rate because topical shards have skewed posting distributions. Shards with long inverted lists could be assigned to the same core, forcing them to compete for cache space. Log-based assignment reduced this competition by spreading popular shards across different cores. We found that log-based shard-to-core assignment enables caching of 10% more vocabulary than with random assignment.

This experiment shows that although using the same caching strategy, the cache performance of topically indexed shards can be worse than that of randomly-partitioned shards due to skewed term distributions across shards. However, a log-based assignment of shards to cores solves this problem and produces cache hit rates comparable to exhaustive search with randomly-organized shards.

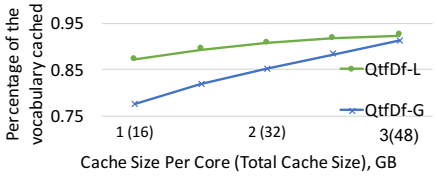

Fig. 2: Percentage of training log vocabulary cached by different caching strategies.

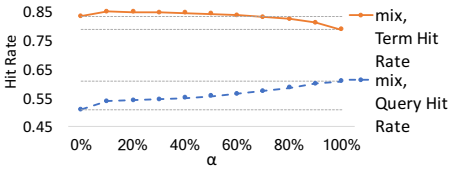

Fig. 3: Hit rate of mixed caching. α: percent of cache used by `QtfDf-G`. Cache size: 2GB.

5.2 Local Term Selection (`QtfDf-L`)

The second experiment explored the use of local term and query distributions to improve caching for *selective* search. It compares the cache hit rates of global (`QtfDf-G`) and local (`QtfDf-L`) strategies. Table 1 shows the hit rates of different strategies.

`QtfDf-L` produced a substantially higher **term hit rate** than `QtfDf-G` for small caches. The local-ranking term hit rate was 18% higher than the global-ranking hit rate for the cache size $m = 1$GB; this gap gradually decreased when cache size increased.

If `QtFDf-G` decides to cache a term such as 'apple', it caches the postings from all shards. `QtfDf-L` may cache the postings from only a few shards (e.g., 'cooking' and 'technology' topics), saving space to cache other terms from other shards. Fig. 2 shows that `QtfDf-L` (partially) caches almost the whole vocabulary, while `QtfDf-G` (fully) caches a smaller vocabulary. This difference is stronger when the cache size is small.

When few shards are searched ($n = 3.9$), the local method `QtfDf-L` has equal or higher **query hit rate** than the global method `QtfDf-G`. However, when the query runs on more shards ($n = 8.5$), `QtfDf-L`'s query hit rate drops substantially. The next two experiments examined this behavior.

5.3 Rank-biased Local Term Selection (`rQtfDf-L`)

One cause of cache misses in `QtfDF-L` is the mismatch between the caching strategy and the resource selection algorithm. Resource selection favors shards that have many documents containing query term t (high $df_s(t)$). However, `QtfDf-L` penalizes high $df_s(t)$, thus t is not always cached for the most-relevant shards.

To align caching strategy with resource selection, we propose a rank-biased strategy, $\texttt{rQtfDf-L}(t,s) = \frac{rqtf_s(t)}{df_s(t)}$, where $rqtf_s(t)$ is a discounted query term frequency based on how resource selection ranks shards. It is defined as $rqtf_s(t) = \sum_{q \in Q_s \& t \in q} 1/\log(r_s^q + 1)$, where r_s^q is the rank of shard s for query q given by resource selection. Each occurrence of term t in the shard's local query log Q_s is discounted by $1/\log(r_s^q + 1)$. Term t is more likely to be cached if the shard ranks highly for queries that contains t.

Table 1 reports the hit rates of `rQtfDf-L`. `rQtfDf-L` significantly improved the query hit rate of `QtfDf-L` for small cache sizes. In the high precision setting, `rQtfDF-L` has the best cache performance. In the high recall setting, it improved `QtfDf-L`'s query hit rate by up to 15%, but is still lower than `QtfDf-G`. For term hit rate, `rQtfDf-L` maintains the performance of `QtfDf-L`; both local methods are significantly better than the global method for small cache sizes.

Table 1: Hit rate of `QtfDf-G`, `QtfDf-L` and `rQtfDf-L`. n: average number of shards searched. † and ⋆ indicate statistically significant improvement over `QtfDf-G` and `QtfDF-L`, respectively. Methods are compared by a pairwise permutation test with $p = 0.05$.

	$n = 3.9$ (High Precision)						$n = 8.5$ (High Recall)					
	Term Hit Rate			Query Hit Rate			Term Hit Rate			Query Hit Rate		
Cache	1GB	2GB	3GB	1GB	2GB	3GB	1GB	2GB	3GB	1GB	2GB	3GB
`QtfDf-G`	0.59	0.78	0.89	0.38	0.61	0.76	0.57	0.79	0.90	**0.38***	**0.60***	**0.76***
`QtfDf-L`	0.68†	0.82†	0.89	0.40	0.61	0.75	**0.67**†	**0.84**†	0.90	0.27	0.49	0.67
`rQtfDf-L`	**0.70**†	**0.85**†	**0.91**	**0.43**†*	**0.63**	**0.77**	0.66†	**0.84**†	**0.91**	0.31*	0.53*	0.68

Table 2: Hit rate of `QtfDf-G`, `rQtfDf-L`, and `mix` caching strategy. †: statistically significant improvement over `QtfDf-G` using pairwise permutation test with $p = 0.05$. ¶: equivalence to `QtfDf-G` using a noninferiority test with 5% margin and 95% confidence interval.

	$n = 3.9$ (High Precision)						$n = 8.5$ (High Recall)					
	Term Hit Rate			Query Hit Rate			Term Hit Rate			Query Hit Rate		
Cache	1GB	2GB	3GB	1GB	2GB	3GB	1GB	2GB	3GB	1GB	2GB	3GB
`QtfDf-G`	0.59	0.78	0.89	0.38	0.61	0.76	0.57	0.79	0.90	**0.38**	**0.60**	**0.76**
`rQtfDf-L`	**0.70**†	0.85†	0.91¶	**0.43**†	0.63¶	0.77¶	**0.66**†	**0.84**†	0.91¶	0.31	0.53	0.68
`mix`	**0.70**†	**0.86**†	**0.92**¶	0.42†	**0.64**¶	**0.78**¶	0.64†	**0.84**†	**0.92**¶	0.36¶	0.58¶	0.74¶

5.4 Mixed Caching

The second reason that local term selection methods have lower query hit rates is that they do not cache *general* terms. Local methods favor terms that characterize the shard. It does not cache terms such as 'main', 'cheap', and 'American' that are not topical, but instead commonly modify the scope of other query terms. These general terms occur in a large portion of the query traffic, causing partial miss for many queries.

The last experiment combined global and local methods by allocating part of the cache to each method. `QtfDf-G` selected terms cached by all shards. `rQtfDf-L` selected terms cached by individual shards. The percentage of cache used by `QtfDf-G` was α.

Fig. 3 shows the hit rates as a function of α. First there is an increase in term hit rate, meaning that the terms with the highest global `QtfDf-G` scores are of high importance and should always be cached. Term hit rate then remains higher than `rQtfDf-L` until 70% of the cache is used by globally-selected terms. Term hit rate is stable because local caching can maintain a useful set of cached terms even with fairly small cache sizes (Fig. 2). On the other hand, query hit rate increased almost linearly as the cache size used by the global method (α) grew. This result suggests that a small local cache is sufficient to maintain term hit rate, whereas a larger global cache is required to provide a good query hit rate. In other words, there is a small vocabulary of important shard-specific terms, and a larger vocabulary of important topic-independent terms.

Table 2 shows hit rates for a mixed strategy that devotes α=70% of the cache to global caching. In the high-precision setting, `mix` slightly improved `rQtfDf-L`; they both outperformed the `QtfDf-G` baseline. In the high-recall setting, the query hit rate for `mix` was almost equal to the `QtfDf-G` baseline, however the term hit rate was improved by over 10% for small cache sizes. Thus, response time remains the same for most users

(as determined by query hit rate), but there is less disk activity and I/O (as determined by term hit rate). A parallel system is likely to have moderately improved throughput.

6 Conclusion

Selective search uses topical index shards and selects just a few shards per query, which skews the distributions of postings and queries across index shards. It has been an open question how typical inverted list caching algorithms perform in this architecture.

Our experiments show that the skewed distribution of postings can cause lower query hit rate in topically indexed shards. However, this effect can be eliminated by using a log-based shard assignment policy to spread popular shards across different cores, which prior research showed also improves load balancing and query throughput.

Global and local term selection methods provide choices to system designers using selective search. Global selection is best for user response time (as determined by query hit rate). Local selection using shard-specific information significantly reduces the posting list I/O (as determined by term hit rate) for small cache sizes, which increases throughput in parallel systems; however the query hit rate is lower compared with the global method when larger number of shards are searched. A rank-biased local variant that incorporates the preferences of resource selection significantly improves the query hit rate for small cache sizes.

A mixed strategy allows trade-offs between user response time and system throughput. Devoting a small portion of the cache to the local method maintains the response time of the global method for most users, but reduces system-wide posting list I/O.

7 Acknowledgments

This research was supported by National Science Foundation grant IIS-1302206. Any opinions, findings, and conclusions in this paper are the authors' and do not necessarily reflect those of the sponsors.

References

[1] Aly, R., Hiemstra, D., Demeester, T.: Taily: Shard selection using the tail of score distributions. In: Proc. SIGIR (2013)
[2] Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F.: The impact of caching on search engines. In: Proc. SIGIR (2007)
[3] Dai, Z., Xiong, C., Callan, J.: Query-biased partitioning for selective search. In: Proc. CIKM (2016)
[4] Kim, Y., Callan, J., Culpepper, J., Alistair, M.: Load-balancing in distributed selective search. In: Proc.SIGIR (2016)
[5] Kulkarni, A.: Efficient and Effective Large-Scale Search. Ph.D. thesis, Carnegie Mellon University (2013)
[6] Marin, M., Gil-Costa, V., Gomez-Pantoja, C.: New caching techniques for web search engines. In: Proc. HPDC (2010)
[7] Saraiva, P.C., Silva de Moura, E., Ziviani, N., Meira, W., Fonseca, R., Riberio-Neto, B.: Rank-preserving two-level caching for scalable search engines. In: Proc. SIGIR (2001)
[8] Zhang, J., Long, X., Suel, T.: Performance of compressed inverted list caching in search engines. In: Proc. WWW (2008)