Query Load Balancing by Caching Search Results in Peer-to-Peer Information Retrieval Networks

Almer S. Tigelaar
a.s.tigelaar@cs.utwente.nl
Database Group, University of Twente

Djoerd Hiemstra
hiemstra@cs.utwente.nl
Database Group, University of Twente

ABSTRACT

For peer-to-peer web search engines it is important to keep the delay between receiving a query and providing search results within an acceptable range for the end user. How to achieve this remains an open challenge. One way to reduce delays is by caching search results for queries and allowing peers to access each others cache. In this paper we explore the limitations of search result caching in large-scale peer-to-peer information retrieval networks by simulating such networks with increasing levels of realism. We find that cache hit ratios of at least thirty-three percent are attainable.

Keywords

distributed query processing, peer-to-peer simulation.

1. INTRODUCTION

In peer-to-peer information retrieval a network of peers provide a search service collaboratively. We define a peer as a computer system connected to the Internet. The term peer refers to the fact that in a peer-to-peer system all peers are considered equal and can both supply and consume resources. In a peer-to-peer network each additional peer adds extra processing capacity and bandwidth in contrast with typical client/server search systems where each additional client puts extra strain on the search server. When such a peer-to-peer network has good load balancing properties it can scale up to handle millions of simultaneous peers. However, the performance of such a network is strongly affected by how well it can deal with the constant and rapid joining and departing of peers which is called *churn*.

We study peer-to-peer information retrieval systems where the collection is split over the peers. Each peer contains a subset of all the documents in the collection, and thus also contains a partial index. Since presumably relevant search results can be located at any peer in the network it is often difficult to route a query to the right peer. This problem is commonly approached by using different network topologies and replication of index data. Indeed, query routing is a difficult problem in peer-to-peer information retrieval [4].

In this paper we explore search result caching as a technique that can be used to both perform load balancing and increase the availability of search results. Instead of forward push-based replication of an index, we use a pull-based caching approach [1]. We experiment with fifty times more peers than any existing scientific peer-to-peer experiments we know of.

We define the following research questions:

1. What fraction of queries can be potentially answered from a cache?
2. How can the cache hit distribution in a peer-to-peer network be characterised?
3. How does churn affect caching?

2. RELATED WORK

Markatos [5] analyses the effectiveness of caching search results for a centralised web search engine combined with a caching web accelerator. Their experiments suggest that one out of three queries submitted has already been submitted previously. They conclude that cache hit ratio's between 25 to 75 percent are possible.

Skobeltsyn and Aberer [7] investigate how search result caching can be used in a peer-to-peer information retrieval network. When a peer issues a query it first looks in a distributed meta-index, kept in a distributed hash table, to see if there are peers with cached results for this query. If so, the results are obtained from one of those peers, but if no cached results exist, the query is broadcast through the entire network. The costs of this fallback are $O(n)$ for a network of n peers. The authors further try to increase the performance of their system by using query subsumption: obtaining search results for subsets of the terms of the full query. They show that with subsumption cache hit rates of 98 percent are possible as opposed to 82 percent without. Interestingly, only 18 percent of the queries in the query log they use appear only once. Perhaps this is because their log is a Wikipedia trace as this is inconsistent with our findings.

3. EXPERIMENTS

3.1 Introduction

Our experiments are intended to give insight into the *maximum benefits* that can be gained by caching. Each experiment has been repeated five times, averages are reported,

Table 1: Query log statistics.

Queries (incl. duplicates)	21,082,980
Users	651,647

no differences between runs were observed that exceeded 0.5 percent. We assume that there are three types of peers: *supplier peers* that have their own locally searchable index, *consumer peers* that issue queries to the network, and *mixed peers* that have both an index and issue queries. We further assume that all peers in our network are willing to cooperate by caching search results. For query routing we introduce a party called the *tracker* which keeps track of which peer can answer what query. The usage of a tracker is inspired by BitTorrent [3]. However, in BitTorrent the tracker is used for locating a specific file: *exact search*, and not for searching to obtain a list of peers which have presumably relevant search results: *approximate search*. In reality the tracker can be implemented in various ways: as a central machine, as a group of high capacity machines in the network, as a distributed hash table or by fully replicating a global data index over all peers. In our experiments we make two important assumptions: firstly, that caches are unbounded in size, and secondly that cached results retain their validity: they need not be invalidated. When dropping either of these two assumptions, caching would become less effective.

3.2 Collection

To simulate a network of peers posing queries we use a large search engine query log [6]. This log consists of over twenty million queries of users recorded over a period of three months. Each unique user in the log is a distinct peer in our experiment. We made several adjustments to it to make our simulations more realistic. Firstly, some queries are censored and appear in the log as a single dash [2]: these were removed. Secondly, we removed results by one user in the log that poses an unusually high number of queries: likely some type of proxy.

Furthermore, we assume that a search session lasts at most one hour. If the exact same query is recorded multiple times in this time window, they are assumed to be requests for subsequent search result pages and thus we use it only once in the simulation. Table 1 shows statistics regarding the log. While the log is sorted by numeric user identifier, for realistic simulation we play back the log in chronological order. We noticed that one day in the log, May 17th 2006, is truncated and does not contain data for the full day, but only for about half an hour after midnight. This has consequences for one of our experiments described later. For clarity: we do not use real search results for the queries in the log. In our experiments we make the assumption that specific subsets of peers have search results and obtain experimental results by counting hits only.

3.3 Centralised

Let us first consider the case where one supplier peer in the system is the only peer that can provide search results. This peer does not pose queries itself. This scenario provides a baseline which resembles a centralised search system. Calculating the query load on the peer-to-peer network is trivial in this case: all 21 million queries *have to be* answered by this single central supplier peer.

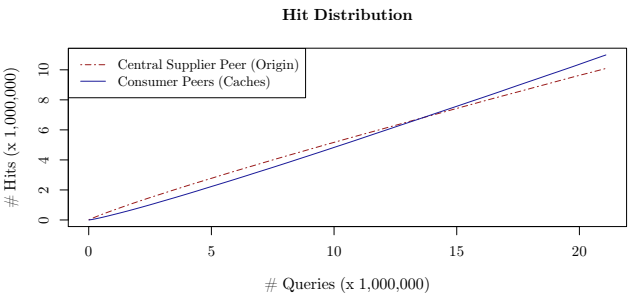

Figure 1: Distribution of hits when peers perform result caching (N=651,647 peers).

However, what if the search results provided by the central supplier peer can be cached by the consuming peers? In this scenario the tracker makes the assumption that all queries can initially be answered by the central supplier peer. However, when a consuming peer asks the tracker for advice for a particular query, this peer is registered at the tracker as caching search results for that query. Subsequent requests for that same query are offloaded to caching peers by the tracker. When there are multiple possible caching peers for a query, one is selected randomly.

Figure 1 shows the number of search results provided by the origin central supplier peer and the summed number of hits on the caches at the consumer peers. It turns out that results for about half of the queries need to be given by the supplier at least once. The other half can be served from the caches of the other peers. Since the maximum achievable cache hit ratio is approximately 0.5, caching can reduce the load on a central peer by about 50 percent. Caching becomes more effective as more queries flow through the system. This is due to the effect that there are increasingly more repeated queries and less unique queries. So, you always see slightly fewer new queries than queries you have already seen as the number of queries increases.

How many results can a peer serve from its local cache and for how many does it have to consult caches at other peers? The local cache hit ratio climbs from around 22 percent for several thousand queries to 39 percent for all 21 million queries. So, the majority of cache hits is on external peers (between 61 and 78 percent).

Let us take a closer look at those external hits. We define a peer's share ratio as follows:

$$shareratio = \#cachehits/\#queries \quad (1)$$

Where *cachehits* is the number of external hits on a peer's cache, meaning: all cache hits that are not queries posed by the peer itself. *Queries* is the number of queries issued by the peer. A *shareratio* of 0 means that a peer's cache is never used for answering external queries, between 0 and 1 means that a peer is sending more queries than it answers, and a ratio above 1 indicates that a peer is actually serving results for more queries than it sends.

Figure 2 shows that about 20 percent of peers does not share anything at all. It turns out that the majority of peers, 68 percent, at least serve results for some queries, whereas only 12 percent, about 80,000 peers, serve results for more queries than they issue.

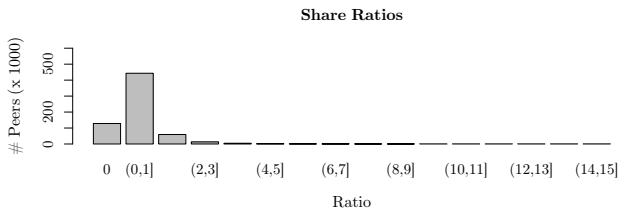

Figure 2: Observed share ratios (N=651,647 peers).

3.4 Decentralised

Now that we have shown the effectiveness of caching for offloading one central peer, we make the scenario more realistic. Instead of a central peer we introduce n peers that are *both* suppliers and consumer at the same time. These mixed peers are chosen at random. They serve search results, pose queries and also participate in caching. The remaining peers are merely consumers that can only cache results.

The central hits in the previous sections become hits per supplier in this scenario. How does the distribution of search results affect the external cache hit ratios of the supplier peers? We examine two distribution cases:

1. For each query there is always only exactly one supplier with unique relevant search results.

2. The number of supplier peers that have relevant search results for a query depends on the query popularity. There is always at least one supplier for a query, but the more popular a query the more suppliers there are (up to all n suppliers for very popular queries).

For simplicity we assume in both cases that there is only one set of search results per query. In the first case this set is present at exactly one supplier peer. However, the second case is more complicated: among the mixed peers we distribute the search results by considering each peer as a bin covering a range in the query frequency histogram. We assume that for each query there is at least one peer with relevant results. However, if a query is more frequent it can be answered by more mixed peers. The most frequent queries can be served by *all* n supplier peers. The distribution of search results is, like the queries themselves, *zipf* over the mixed peers. We believe that this is realistic, since popular queries on the Internet tend to have many search results as well. In this case the random choice is between a variable number m of n peers that supply search results for a given query. Thus, when the tracker receives a query for which there are multiple possible peers with results it chooses one randomly.

We performed two experiments to examine the influence on query load. The first is based on case 1, where there is always one supplier given an input query. The second is based on case 2 where the number of suppliers varies per query. For case 2 we first used the query log to determine the popularity of queries and then used this to generate the initial distribution of search results over the suppliers. This distribution is performed by randomly assigning the search results to a fraction of the suppliers depending on the query popularity. Since normally the query popularity can only be approximated, the results represent an ideal outcome.

Table 2: Original search results and cache hits (21,082,980 queries; 651,647 peers of which 10,000 are suppliers). All suppliers operate in mixed mode.

	Case 1	Case 2
Suppliers (origin)	11,599,060	12,110,592
Consumers internal (caches)	3,682,995	3,930,025
Consumers external (caches)	5,800,925	5,042,363

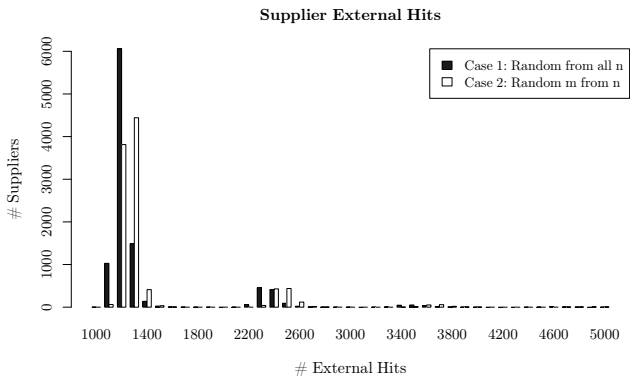

Figure 3: Supplier external hit distributions (N=651,647 peers, n=10,000 suppliers).

We used n = 10,000 supplier peers in a network of 651,647 peers in total (about 1.53 percent). This mimics the Internet which has a small number of websites compared to a very large number of surfing clients.

Figure 3 and Table 2 show the results. The number of original search results provided by the suppliers is about five percent higher than in the central peer scenario. This is the combined effect of no explicit offloading of the supplier peers by the tracker, and participation of the suppliers in caching for other queries. In the second case there is slightly more load on the supplier peers than in the first case: 57 percent versus 55 percent. The hit distribution in Figure 3 is similar even though the underlying assumptions are different. About 87 percent of peers answer between 1000 and 1500 queries. A very small number of peers answers up to about five times that many queries. Differences are found near the low end, which seems somewhat more spread in the first than in the second case. Nevertheless, all these differences are relatively small. The distribution follows a wave-like pattern with increasingly smaller peaks: near 1300, 2500, 3700 and 4900. The cause of this is unknown.

3.5 Churn

The experiments thus far have shown the maximum improvements that are attainable with caching. In this section we add one more level of realism: we no longer assume that peers are on-line infinitely. We base this experiment on case 1 above where the search results are uniformly distributed over the suppliers. The query log contains timestamps and we assume that if a specific user has not issued a query for some period of time, that his session has ended and its cache is temporarily no longer available. If the same user issues a query later on (comes back on-line), its cache becomes available again. This simulates churn in a peer-to-peer network

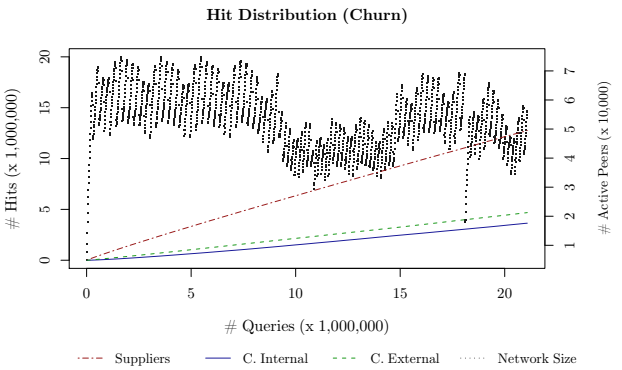

Figure 4: Distribution of hits under churn conditions (N=651,647 peers).

where peers join and depart from the network. All peers, including supplier peers, are subject to churn. For bootstrapping: if there are no suppliers on-line at all, an off-line one is randomly chosen to provide search results.

Assuming that all peers are on-line for a fixed amount of time is unrealistic. Stutzbach and Rejaie [8] show that download session lengths, post-download lingering time and the total up-time of peers in peer-to-peer file sharing networks are best modelled by using *Weibull distributions*. However, our scenario differs from file sharing. An information retrieval session does not end when a search result has been obtained, rather it spans multiple queries over some length of time. Even when a search session ends, the machine itself is usually not immediately turned off or disconnected from the Internet. This leads us to two important factors for estimating how long peers remain joined to the network. Firstly, there should be some reasonable minimum that covers at least a browsing session. Secondly, up-time should be used rather than 'download' session length. As soon as a peer issues its first query we calculate the remaining up-time of that peer in seconds as follows :

$$remaininguptime = 900 + (3600 \cdot 8) \cdot w \qquad (2)$$

where w is a random number drawn from a Weibull distribution with $\lambda = 2$ and $k = 1$. The w parameter is usually near 0 and very rarely near 10. The up-time thus spans from at least 15 minutes to at most about 80 hours. About 20 percent of the peers is on-line for longer than one day. This mimics the distribution of up-times as reported in [8].

Figure 4 shows the results: the number of origin search results served by suppliers as well as the number of internal and external hits on the caches of consumer peers. We see that the number of supplier hits increases to over 12.75 million: over 1.16 million more compared to the situation with no churn. The majority of this increase can be attributed to a decrease in the number of external cache hits. The dotted cloud shows the size of the peer-to-peer network on the right axis: this is the number of peers that is on-line simultaneously. We can see that this varies somewhere between about 30,000 and 80,000 peers. There is a dip in the graph caused by the earlier described log truncation.

4. CONCLUSION

We conducted several experiments that simulate a large-scale peer-to-peer information retrieval network. Our research questions can be answered as follows:

1. At least 50 percent of the queries can be answered from search result caches in a centralised scenario. This drops to 45 percent for the decentralised case.
2. Share ratios are skewed which suggests that additional mechanisms are needed for cache load balancing.
3. Introducing churn into a peer-to-peer network reduces the maximum cache hits by 12 percent to 33 percent.

We have shown the potential of caching under increasingly realistic conditions. Caching search results significantly offloads the origin suppliers that provide search results under all considered scenarios. This could be even further improved by applying query subsumption, term re-ordering and stemming. These techniques may decrease the quality of the search results, but also offer more effective usage of caches. This is needed when extra layers of realism are added to the experiments by working with individual search results instead of result sets, by experimenting with finite size caches, and by invalidating cached results over time. It would be useful to experiment with a combination of caching *and* replication. Finally, much work remains to be done in peer-to-peer information retrieval, especially in investigating the properties that hold in large-scale simulations.

5. ACKNOWLEDGEMENTS

We wish to thank Dolf Trieschnigg. This paper was created using only Free and Open Source Software. We gratefully acknowledge the support of the Netherlands Organisation for Scientific Research (NWO) under project 639.022.809.

References

[1] Baentsch, M., Baum, L., Molter, G., Rothkugel, S., and Sturm, P. 1997. Enhancing the web's infrastructure. *Internet Computing 1,* 2 (Mar.), 18–27.

[2] Brenes, D. J. and Gayo-Avello, D. 2009. Stratified analysis of aol query log. *Information Sciences 179,* 12, 1844 – 1858.

[3] Cohen, B. 2003. Incentives build robustness in bittorrent. In *Proceedings of P2PEcon.*

[4] Lu, J. and Callan, J. 2006. Full-text federated search of text-based digital libraries in peer-to-peer networks. *Information Retrieval 9,* 4, 477–498.

[5] Markatos, E. P. 2001. On caching search engine query results. *Computer Communications 24,* 2 (Feb.), 137–143.

[6] Pass, G., Chowdhury, A., and Torgeson, C. 2006. A picture of search. In *Proceedings of InfoScale.* Hong Kong, 1.

[7] Skobeltsyn, G. and Aberer, K. 2006. Distributed cache table: efficient query-driven processing of multi-term queries in p2p networks. In *Proceedings of P2PIR.* Arlington, Virginia, US, 33–40.

[8] Stutzbach, D. and Rejaie, R. 2006. Understanding churn in peer-to-peer networks. In *Proceedings of IMC.* Rio de Janeiro, BR, 189–202.