Locaware: Index Caching in Unstructured P2P-file Sharing Systems

Manal El Dick, Esther Pacitti

Locaware: Index Caching in Unstructured P2P-file Sharing Systems

Manal El Dick
Atlas Team, INRIA and LINA, University of
Nantes, France
manal.el-dick@univ-nantes.fr

Esther Pacitti
Atlas Team, INRIA and LINA, University of
Nantes, France
esther.pacitti@univ-nantes.fr

ABSTRACT

Though widely deployed for file-sharing, unstructured P2P systems aggressively exploit network resources as they grow in popularity. The P2P traffic is the leading consumer of bandwidth, mainly due to search inefficiency, as well as to large data transfers over long distances. This critical issue may compromise the benefits of such systems by drastically limiting their scalability. In order to reduce the P2P redundant traffic, we propose Locaware, which performs index caching while supporting keyword search. Locaware aims at reducing the network load by directing queries to available nearby results. For this purpose, Locaware leverages natural file replication and uses topological information in terms of file physical distribution.

1. INTRODUCTION

Despite the emergence of sophisticated overlay structures, unstructured peer-to-peer (P2P) systems remain highly popular and widely deployed. They exhibit many simple yet attractive features, such as low-cost maintenance and high flexibility in data placement and node neighborhood. Unstructured P2P systems are particularly used in file-sharing communities due to their capacity of handling keyword queries, i.e., queries using some keywords instead of the whole filename. Unfortunately, as these systems grow in popularity, they aggressively exploit network resources, typically by consuming huge amounts of bandwidth [14]. This issue leads inevitably to performance deterioration and user's disatisfaction.

One principal cause of the network load is search inefficiency. Blind mechanisms, which are commonly employed in unstructured P2P networks, induce a heavy and redundant traffic. In fact, inefficient searches cause unnecessary messages that overload the network, while missing requested files.
Several analyses [11, 15] found the P2P file-sharing traffic highly repetitive because of the temporal locality of queries. They actually observed that most queries request a few popular files and advocated the potential of caching query responses to answer subsequent queries without flooding over the entire network: a query response, which holds information about the location of the requested file, can be cached in the form of index on its way back

to the requestor. However, the same results can be unnecessarily cached among neighbors. Therefore, index caching should be carefully examined to avoid redundancy and storage overhead.

Another important factor that aggravates the network load, is the large data transfer over long distances. A query can be directed to a copy of the desired file which is hosted by a physically distant peer, while other copies may be available at closer peers. Hence, file download can consume a significant amount of bandwidth and thereby overload the network. In addition, the user experience dramatically degrades due to the relatively high latency perceived. In P2P file sharing, a peer that requested and downloaded a file, can provide its copy for subsequent queries. As a consequence, popular files, which are frequently requested, become naturally well-replicated [4]. Hence, search techniques should leverage the natural file replication to efficiently select results.

In order to address the issues identified above, we make the following contributions:

- We propose Locaware, a location-aware index caching technique that aims at reducing the P2P unnecessary bandwidth consumption with minimum storage overhead. A peer intercepts query responses and selectively caches several indexes per file, along with information about their physical locations. As a consequence, a peer answers a query by providing several possibilities. This approach aims at finding a nearby copy to optimize file transfer, while it improves the file availability probability.

- We propose a query routing strategy that supports keyword queries and avoids flooding overhead. For this purpose, Bloom filters are used to express keywords of filenames cached at peers, and are then propagated to neighbors. Thus, a peer routes a query by querying its neighbors' Bloom filters.

- We evaluated the performance of our solution through simulation using PeerSim, and showed the effectiveness of Locaware in reducing the search traffic in unstructured systems.

The rest of this paper is organized as follows. Section 2 reviews the different kinds of approaches that aim at reducing the unnecessary P2P traffic, as related work. Section 3 introduces the background of our work and defines the main concepts on which Locaware relies. Section 4 describes in details our contributions. Section 5 evaluates the performance of our approach through simulation. In section 6, we present our conclusions.

2. RELATED WORK

Many studies have focused on reducing P2P traffic in unstructured systems, while different types of approaches have been exploited. Our work mainly falls into three categories: search-based, caching-based and topology-based.

Search-based approaches [5, 10], basically consist of maintaining information about neighbors, and using them to route queries. A peer can either hold summarized lists of files stored at its neighbors or statistics based on previous searches (e.g. number of results returned from each neighbor). Each time the peer needs to forward a query, it selects a subset of its neighbors according to the maintained information. These techniques do not incorporate location awareness, typically by directing a query to a physically distant file, which consumes high bandwidth at transfer and contributes to increasing the traffic.

Caching-based approaches aim at caching content (e.g. files) or indexes (e.g. file locations) to limit the extent of flooding, when searching for some content. Centralized caching [12] at the gateway of an organization does not exploit peers resources and is likely to produce bottlenecks. The authors in [15] propose that each peer caches all passing query responses, which results in large amount of duplicated and redundant cached results among neighboring peers. In Dicas [16], query responses are cached in specific groups of peers based on a specific hashing of the filenames. Guided by the predefined hashing, queries are then routed towards peers which are likely to have the desired indexes. However, Dicas is not optimized for keyword searches which are the most common in the context of P2P file sharing: some proposed strategy consists in caching indexes based on hashing query keywords instead of the whole filename, which causes a large amount of duplicated cached indexes. Moreover, Dicas does not exploit location awareness nor the well-replication of popular files.

Topology-based approaches [9, 13] focus on optimizing the P2P overlay topology by exploiting information about the underlying network. The goal is to construct an overlay that reflects the underlying topology. This category of solutions do not deal with search efficiency and thus can be considered as comlementary to our work.

3. PROBLEM DEFINITION

In this section, we introduce the main terms and concepts in order to define the problem. First, we briefly describe unstructured P2P file-sharing systems. Second, we present index caching and finally the problem statement.

3.1   P2P File-Sharing Systems

In unstructured P2P networks such as Gnutella, each peer joins the network by establishing logical links to randomly chosen peers, referred to as its neighbors. Normally, the neighborhood of a peer is set without knowledge of the underlying topology. Participant peers are highly dynamic and autonomous, failing or leaving the network at any moment.

In such P2P systems, peers share files of any type specified by the application. We use f to refer to both the file object and its filename, as the meaning will be clear from the context. Filenames are broken into keywords following predefined rules. Peers request files by submitting queries to the P2P network. A query q is commonly expressed by some keywords related to the queried filename instead of the whole filename.

Query routing is done by blindly flooding q over the P2P network

and is bounded by a fixed TTL. Query responses qr follow the reverse path of their corresponding q, back to the requesting peer: q can be satisfied by any file f which filename contains all keywords of q. Thus, a query response qr_f contains the filename and the location of a copy of f, which refers to the IP address of a peer n located by the query and providing f (peer n is noted p_f). The requesting peer downloads f via direct connection and then becomes a provider peer p_f for subsequent requests for f. If a file f is requested frequently, then as more requesting peers download f, there will be many copies of f and thereby many providers p_f within the system.

3.2   Index caching

Index caching consists of caching a query response qr, in the form of a file index, by peers along qr path. An index of f contains the filename and the IP address of some provider peer p_f. Therefore, each peer n maintains a cache of file indexes called *response index* and noted RI_n.

In Dicas [16], each peer n randomly chooses a group Id noted Gid_n ($Gid_n \in [0\,..\,M-1]$ with M a system parameter). Gid_n *matches* a filename f if the following condition is satisfied:

$$Gid_n = \text{hash}(f) \bmod M \qquad (1)$$

Group Ids are used to restrict index caching in some peers along the query reverse path, in order to avoid redundant indexes among neighbors. Hence, a query response qr_f is only cached in RI of peers with matching Gid wrt. f.

3.3   Problem Statement

We formally define our problem as follows. Given an unstructured P2P file-sharing system, let:

- **N** be the set of all participant peers n, such that each peer n is assigned a Gid.

  - $\forall n \in N$; RI_n caches query responses qr_f such that $\text{hash}(f) \bmod M = Gid_n$.

- **PF** be the set of popularly shared files f, such that each f may be provided by multiple provider peers $p_f \in \mathbf{N}$.

  - $\forall f \in \mathbf{PF}$; filename $f = \{kw_i; 0 \le i < K\}$
    * kw_i is a keyword contained in filename f
    * K is the total number of keywords contained in filename f

- **PQ** be the set of common queries q which request files of **PF** and are expressed as follows:

  - $\forall q \in \mathbf{PQ}$; $q = \{kw_i \in f; 0 \le i < X\}$ and $1 \le X \le K$
    * X is a random number of keywords from filename f

Our objectives are the following:

- perform index caching at peers $n \in \mathbf{N}$ whith storage efficiency.

- route q efficiently towards peers with indexes of f.

- exploit replication of f, i.e., several p_f as well as their physical locations and incorporate this information in RI_n.
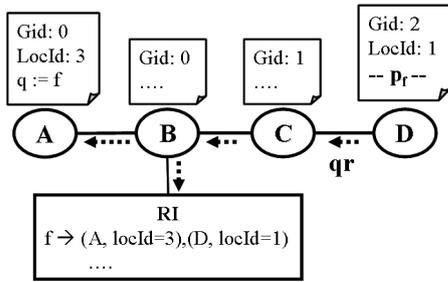
Figure 1: Index caching in Locaware

4. LOCAWARE: EFFICIENT CACHING AND SEARCHING IN P2P FILE-SHARING

In this section, we present Locaware, our approach to limit the wasted bandwidth in unstructured P2P systems. We describe first the location-aware index caching strategy and second the query searching solution which uses our location-aware index and supports keyword queries.

4.1 Location-aware Index Caching

4.1.1 Physical locations

We introduce location awareness in response indexes, in terms of information about the physical location of the file providers. For this purpose, we assume that participant peers can be grouped based on their physical locations. To model these physical locations, we use a common technique used in other works such as [6, 13], which relies on the existence of a set of well-known machines spread across the Internet, called *landmarks*. A peer n can estimate its distance, i.e., its round-trip time (RTT) to each landmark. An ordering of the set by increasing RTT reflects the physical location of peer n. Thus, physically close peers are likely to produce the same ordering. We thereby associate to each possible ordering a location Id noted $locId$. At its arrival, each peer computes its own locId based on its RTT measurements.

Thus, we can construct a location-aware response index, where each index contains, in addition to the IP address of some p_f, the $locId$ of p_f.

4.1.2 Leveraging file replication

We exploit file replication, based on the fact that a peer which has recently requested a file f is likely to have it and can thereby serve subsequent requests for f. Hence, a peer that submits a query q for f, will be considered as a new provider p_f by the peers that intercept a response qr_f on its way back to the requesting peer.

The example in Fig. 1 illustrates our caching strategy (query routing details in the next section). In the example, peer A with $locId = 3$, has submitted the query q for file f; peer D with $locId = 1$, has been located because it is a p_f, i.e., a provider for f. Thus, the query response qr_f holds the information about peer D as well as peer A to be considered as a new provider p_f. We suppose that filename f matches peers with $Gid = 0$. Thus, peer B with $Gid = 0$ caches qr_f in its RI.

In consequence, the response index of a peer n may hold for some cached filename f, several provider addresses $p_{f,i}$ and their locIds. Hence, n tries to answer subsequent queries that can be satisfied by f, according to the locId of the querying peer. Suppose that a subsequent query q' for f reaches peer B of the previous example. q' is submitted by peer E with $locId = 1$. Thus, peer B sends back a response qr that contains the entry $(D, 1)$ corresponding to the locId of E, the query originator. The query response also includes IP addresses of some other providers of f with their associated LocIds (e.g. entry $(A, 1)$), to guarantee that E will find an available copy of f with minimum bandwidth requirements. Peer B then adds in its RI the entry $(E, 1)$ as a new provider of f.

Caching multiple indexes per file may lead to an extra storage requirement. However, each peer can control its cache size in function of its storage capacity. Given the high dynamicity of peers, studies [11] in Gnutella showed that cached objects should be kept for a small amount of time to avoid sending stale responses. Thus, peer n constantly updates the list of providers of f in its RI_n as new queries for f pass by n: the most recent p_f entries replace the oldest ones.

4.2 Searching with Keyword Queries Support

In Dicas [16], group Ids are used to route a query q, towards peers that have a high probability of caching indexes satisfying q. A query q is thus sent to neighbors with a matching Gid wrt. q. However, keyword search is not efficient in Dicas, generating redundancy in terms of storage requirements [7]. Thus, we use a Bloom filter to express filenames' keywords in a response index and to send the filter to neighbors.

A Bloom filter [3, 8] is a simple space-efficient data structure for representing a set of elements, in order to support membership queries. When querying a Bloom filter, it never returns false negatives; but it may lead a false positive when it suggests that an element belongs to the set even though it does not.

Each peer n maintains a Bloom filter, noted BF_n, that represents the set of keywords of all cached filenames in RI_n. Whenever n overhears a response qr_f such that f matches Gid_n, n caches qr_f in RI_n, and then inserts each keyword kw_i of f as an element of BF_n.

Neighboring peers exchange their group Ids as well as their Bloom filters. Thus, peer n stores its direct neighbors' Gid and BF. To forward a query q, peer n queries first its neighbors' BF: n checks for each stored BF_i if it matches q, i.e., $\forall kw_i \in q$, kw_i is member of BF_i. Then, n sends q to neighbors with matched BF wrt. q. If no such neighbors is found, query q is sent to neighbors with matched Gid wrt. q or to a highly connected neighbor as a last resort, to avoid blocking the query forwarding. The query is propagated until a satisfying file is found at some node, either in its file storage or in its response index.

A Bloom filter BF_n is built incrementally as new filenames are inserted in RI_n and existing ones discarded. Copies of BF_n held by neighbors of peer n must reflect the content of RI_n, thus n periodically propagates updates of BF_n to neighbors. These small messages do not consume much bandwidth and can be sent along with any data exchange between neighbors. In fact, when a filename is added or deleted, a small number of bits may change in the bit vector of the BF. Thus, n only needs to transmit the location of the changed bits[1].

---

[1]The number of changed bits in a 1200-bit vector of the BF, is limited by 12 at most and the location of each bit by 11 bits. Thus, the information to be sent is limited by $I = 12 * 11 \, bits = 0.132 \, Kb$

5. PERFORMANCE EVALUATION

In this section, we study the gains of Locaware and quantify its trade-off by comparing it to other related approaches. Below, we first describe our experimental methodology and then present our experimental results.

5.1 Experimental Methodology

We evaluate the performance of our proposed solution by simulation over PeerSim [2], a simulator specifically tailored for P2P protocols. Using PeerSim, we generate an unstructured P2P topology of 1000 peers with an average connectivity degree of 3. PeerSim has an event-driven framework that enables us to model the latency of each individual link; however, it does not provide support for simulating bandwidth and CPU resources. To best simulate the P2P environment, we generate an underlying topology of peers connected with links of variable latencies; the model inspired by BRITE [1] assigns latencies between 10 and 500 ms. To implement locations, we use 4 landmarks, which results in 24 possible locIds, because a larger number of landmarks will scatter the peers into many different localities. For instance, given 5 landmarks, i.e., 120 locIds, we only obtain an average of 8 peers with the same locId. In consequence, it would be quite difficult to find for a given requestor peer, a provider with the same locId. Thus, a small number of landmarks is well-adapted to our simulation model. Furthermore, we adjust the provider selection strategy as follows: when a requestor peer does not find a provider with matching locId amongst its received indexes, it measures its RTT to the set of available providers and chooses the one with the smallest RTT.

We suppose that each peer initially shares 3 files, randomly chosen from a pool of 3000, while each filename is formed of 3 keywords, randomly chosen from a pool of 9000. Queries are generated according to Zipf distribution, at the rate of 0.00083 queries per second per peer. To express each query, we randomly choose 1 to 3 keywords from the queried filename. A query search is bounded by a TTL equal to 7. Considering an enlarged response index with 50 filenames of 3 keywords, we set the Bloom Filter size to 1200 bits, which can provide an optimal representation with a negligeable amount of memory.

We compare Locaware to 3 approaches: Flooding, Dicas [16] (designed for filename search) and Dicas-Keys [16] (designed for keyword search). We measure the effectiveness of Locaware by using 3 performance metrics: download distance, search traffic and success rate. We study the effect of the number of queries on each one of these metrics.

5.2 Experimental Results

In the first experiment, we measure the download distance, i.e., the average network distance, in terms of latency, from the requestor peer to the chosen provider peer. The average download distance indicates how well the system exploits the location awareness when selecting a provider for file download. As shown in Fig. 2, the average download distance is decreased by about 14% compared to the other approaches. Also, an interesting observation that could be derived from this experiment is that Locaware shows improvement with the increase of queries, unlike the other approaches which remain stable. This is because Locaware leverages the natural file replication to serve queries from close-by providers. In fact, as more queries for a particular file are generated and served, there will be more providers of this file available in different physical locations. Eventually, a query for this file is more likely to find a provider within the same locality as the requestor. Furthermore,
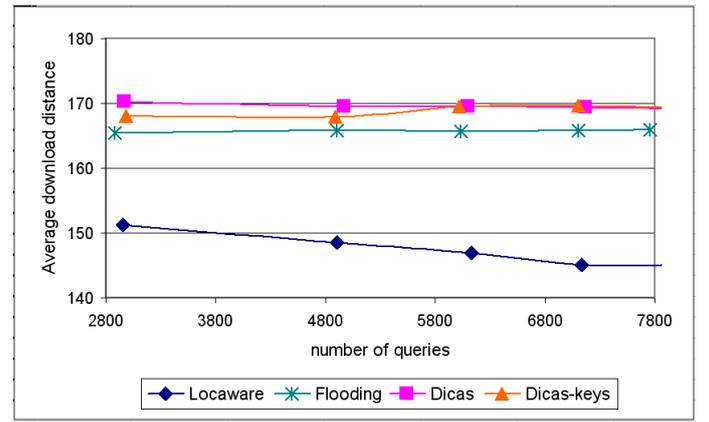

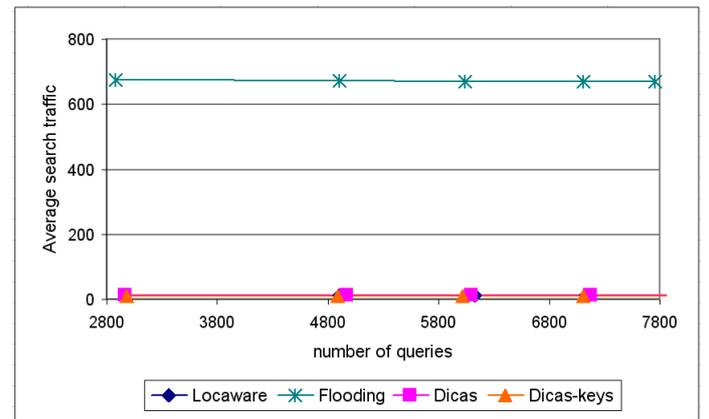**Figure 2: Comparison of download distance.**

Figure 3: Comparison of search traffic.

the improvement would be more significant if the location awareness was also incorporated in the query routing, to improve the efficiency of finding close-by providers.

The second experiment evaluates the search traffic which can be measured as the total number of messages produced by a query in the P2P network. Figure 3 shows that Locaware like Dicas approaches, outperforms flooding by 98% in terms of search traffic reduction. Thus, Locaware preserves the main objective of index caching by drastically limiting the search overhead

Finally, we evaluate the tradeoffs of Locaware against its benefits. For this purpose, we measure how much locaware looses in terms of succes rate, i.e., the rate of queries successfully satisfied to all submitted queries. On the one hand, as expected and shown in Figure 4, flooding outperforms all other techniques because it provides each query with a large search scope and thus avoids missing some results at the cost of a huge traffic overhead. On the other hand, Locaware offers a substantial compensation over Dicas success rate: it increases hit ratio by 23 % wrt. Dicas and 33% wrt. Dicas-keys. Actually, Locaware achieves this improvement because of two features. Firstly, the response index in Locaware has for each file more possibilities of providers than in Dicas and Dicas-keys. Secondly, Locawre provides an efficient support for keyword queries that avoids missing results held by neighbors while Dicas relies on the Gid-based routing that misleads keyword queries.
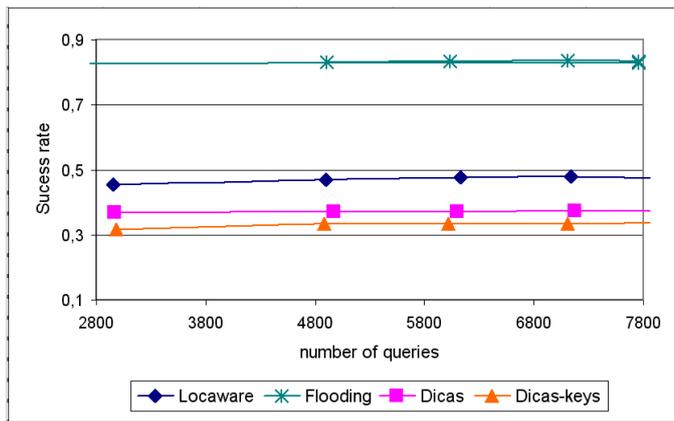
Figure 4: Comparison of success rate.

6. CONCLUSION

Our work falls into the context of P2P file sharing in unstructured systems, and deals with search efficiency in order to reduce the P2P bandwidth consumption. In this paper, we identified limitations of existing solutions. Then, we proposed a solution, Locaware, that leverages typical properties of P2P-file sharing environments such as file replication, as well as useful information about the underlying topology. To efficiently route queries towards desired results, our approach caches file indexes in groups of peers based on the filenames, while efficiently supporting keyword searches. In addition, Locaware aims at improving the probability of finding nearby copies of requested files. Through simulation, we showed that Locaware can significantly limit wasted bandwidth and reduce network resource usage. Results motivate us to elaborate more on location awareness, in order to achieve greater performance improvement. One way is to investigate location-aware query routing in unstructured systems, which has not been fully exploited yet. We also intend to explore P2P structured systems for novel and sophisticated search and caching techniques.

7. REFERENCES

[1] Brite topology generator. http://www.cs.bu.edu/brite/.

[2] Peersim P2P simulator. http://peersim.sourceforge.net.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[4] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making Gnutella-like P2P systems scalable. In *ACM SIGCOMM*, 2003.

[5] Arturo Crespo and Hector Garcia-Molina. Routing indices for P2P systems. In *ICDCS*, 2002.

[6] Manal El Dick, Vidal Martins, and Esther Pacitti. A topology-aware approach for distributed data reconciliation in P2P networks. In *Euro-Par*, 2007.

[7] Manal El Dick, Esther Pacitti, and Patrick Valduriez. Location-aware index caching and searching for P2P systems. To appear in *DBISP2P*, 2007.

[8] Li Fan, Pei Cao, Jussara M. Almeida, and Andrei Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *SIGCOMM*, 1998.

[9] Yunhao Liu, Li Xiao, Xiaomei Liu, Lionel M. Ni, and Xiaodong Zhang. Location awareness in unstructured peer-to-peer systems. *IEEE Trans. Parallel Distrib. Syst.*, 16(2):163–174, 2005.

[10] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and replication in unstructured peer-to-peer networks. In *SIGMETRICS*, 2002.

[11] Evangelos P. Markatos. Tracing a large-scale P2P system: An hour in the life of Gnutella. In *IEEE/ACM CCGrid*, 2002.

[12] Sunil Patro and Y. Charlie Hu. Transparent query caching in peer-to-peer overlay networks. In *IPDPS*, 2003.

[13] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *INFOCOM*, 2002.

[14] Stefan Saroiu, P. Krishna Gummadi, Richard J. Dunn, Steven D. Gribble, and Henry M. Levy. An analysis of internet content delivery systems. In *OSDI*, 2002.

[15] Kunwadee Sripanidkulchai. The popularity of gnutella queries and its implication on scaling. http://www.cs.cmu.edu/ kun-wadee/research/p2p/gnutella.html.

[16] Chen Wang, Li Xiao, Yunhao Liu, and Pei Zheng. Dicas: An efficient distributed caching mechanism for P2P systems. *IEEE Trans. Parallel Distrib. Syst.*, 17(10):1097–1109, 2006.