Evaluation Study of a Distributed Caching Based on Query Similarity in a P2P Network

Mouna Kacimi *
Max-Planck Institut fur Informatik
66123 Saarbrucken, Germany
mkacimi@mpi-inf.mpg.de

Kokou Yetongnon †
Laboratoire LE2I, University of Bourgogne
Dijon 21078 Cedex France
kokou.yetongnon@u-bourgogne.fr

ABSTRACT
Several caching techniques have been used to reduce the bandwidth consumption and to provide faster answers in P2P systems. In this paper, we address the problem of reducing unnecessary traffic in the Hybrid Overlay Network (HON), which consists in organizing peers and data in an n-dimensional feature space for efficient similarity search. We propose a distributed caching schema that group similar queries to increase the success hit and avoid redundancy. We show through extensive simulations that caching in HON decreases significantly the query scope improving search performance.

General Terms
Measurement, Performance

Keywords
P2P Networks, Similarity Search, Caching

1. INTRODUCTION
P2P search has the potential to enhance large-scale IR, reliability and fault tolerance since it does not rely on any centralized resource. Moreover, it offers the possibility to exploit local content of any peer in the network breaking information monopolies. Thus, the design of an efficient P2P search technique raises important issues such as finding the appropriate answers for a given query, optimizing the search cost by reducing the network traffic, and dealing with peers' availability and autonomy.

Many search algorithms have been proposed for P2P systems based on their overlay infrastructure that can be *unstructured* or *structured*. The search in unstructured networks such as Gnutella [5] and KaZaA [8] is based on flooding, where each peer broadcasts the received query to directly connected peers. A Time-To-Live (TTL) mechanism or a random walk method [1, 3] can be used to reduce the number of peers that are involved in processing a query and avoid overloading the network. Flooding techniques are efficient for locating popular data objects for which several duplicate copies exist in a large number of peers. On the other hand, they can exhibit search quality and cost performance for remote unpopular objects which may not be found if TTL limit is reached, or may incur a high search cost if they are found.

In structured P2P systems, search methods use peer content localization information to direct queries to the appropriate peers. DHT (Distributed Hash Table) techniques have been used in several structured systems [9, 13, 15]. They organize data in a key space for efficient data access. Unique identifiers are assigned to the peers and the data. A data object is mapped to the peer with the closest identifier. Each peer maintains a routing table composed of its neighbors' identifiers. A lookup query is routed to and processed by the peer that contains the corresponding data keys. DHT techniques improves the search efficiency by providing a deterministic routing and a high recall. However, the tight control of the network by mapping data indexes to peers requires a high maintenance cost in a highly dynamic network.

Many efforts have been made to avoid large volume of unnecessary traffic in unstructured networks, and to balance the tradeoff between the indexing cost and searching in structured networks. In this paper, we focus on caching strategies that have been a crucial part of these efforts. Various caching techniques based on web proxies have been proposed in the literature [2, 4, 12]. They consist in storing query results in web proxies for future use to reduce response delays and network congestion. Similar techniques have been suggested for P2P systems. However, caching query results in P2P systems is in some way different from caching in web proxy systems. The main difference is due to the fact that in traditional web caching the stored data is kept on well identified static web servers, whereas in P2P systems query results are combination of partial results issued from one or more peers which can frequently connect to and disconnect from the network.

We have proposed in [7] a distributed caching schema to improve the performance of information retrieval in HON [6], a Hybrid Overlay Network for similarity search in a

*Dr. Kacimi did this work during her PhD at the University of Bourgogne in France.

†Professor Yetongnon is the supervisor of this research.

P2P system. HON organizes both peers and data in an n-dimensional feature space based on content description. It is based on two key ideas. First it organizes peers sharing similar contents in the n-dimensional feature space to limit flooding overhead and send queries only to relevant peers. Second, it organizes and places similar data objects in relatively dense and adjacent regions of the feature space to achieve efficient processing of complex queries such as range and nearest neighbor queries. The feature space represents particular attributes associated with data objects (e.g., color for an image, concept or keyword for text document) and is partitioned into cells obtained by dividing the range values of each feature into a number of intervals. Two data are similar if they are mapped to the same cell. The distribution of data objects over the cells defines the similarity between peers. Two peers are similar if their contents are distributed on the same sub regions of the feature space.

Caching in HON consists in storing information about queries descriptions and peers providing relevant answers. The main idea is to assign a cache to each non empty cell of the feature space. Queries that are mapped to the same cell are stored and served from the same cache. Thus, similar queries are always grouped in the same cache, which increases the success rate and avoids redundancy. The focus of this paper is on evaluating the caching performance in HON, studying different replacement policies, and analyzing the impact of caching on the query scope and the search performance. The contributions of this work are three-fold:

1. we study the impact of caching on the search performance in HON showing its efficiency in reducing the query scope and providing a high success hit.
2. we evaluate the behavior of the cache using two different query distributions, namely, uniform and zipfian.
3. we investigate two replacement policies: LRU and NFU, and we analyze their impact on the recall. Moreover, we study their adaptability to dynamic change of peers' content.

The remainder of the paper is organized as follows. In the next section, we give an overview about some P2P caching approaches. In section 3, we give a brief description of HON. Section 4 presents the distributed caching schema proposed in HON. Section 5 describes the cache management policies. Section 6 presents the evaluation results. And finally section 7 concludes the paper.

2. RELATED WORK

P2P caching techniques proposed in the literature can be classified into three approaches. In the first approach, a cache is placed in every peer. Each peer stores in its cache query strings and the results that pass through it. Sripanidkulchai et al. have studied in [14] the popularity of queries in Gnutella which follows a Zipf-like. Markatos et al. in [10] have also studied the traffic of Gnutella and shown that queries tend to be frequently and repeatedly submitted. Moreover, Gnutella peers join and leave the network very frequently. Thus, the query responses may become out-of-date very quickly. To address this problem, the authors proposed a Gnutella caching mechanism that caches query response for only a small amount of time. The effectiveness of Gnutella caching has been studied in both [14] and [10]. Wierzbicki et al. [18] have also evaluated different cache replacement policies that were successful for web traffic, and have introduced new specific policies for FastTrack traffic in KaZaA network.

The second approach is to use a centralized server to cache data. In [11], Patro and Hu have shown that caching at the gateway of an organization can be far more effective than caching at individual peers behind the gateway because it can keep the query traces of all peers. In this manner, if a query Q is issued, it is more likely to find similar queries stored in the cache.

In the third approach, the cache is distributed among selected peers. Wang et al. have analyzed in [16] the Uniform Index Caching (UIC) that stores query results in the peers along the returning path. The experiments show that UIC causes large duplicated and unnecessary cache results among neighboring peers. Therefore, Wang et al. [16] have attempted to cache the responses in some selected peers. The distributed caching DiCAS proposed by [16] is based on hashing. Peers are organized into M groups, where each has a group ID. When a query is issued, its hash code is calculated as follows: query ID=hash (query) Mod M. The query and the corresponding results are sent to the peers with a group ID equal to the query ID. The experiments results presented in [17] show that DiCAS protocol can significantly reduce the network search traffic.

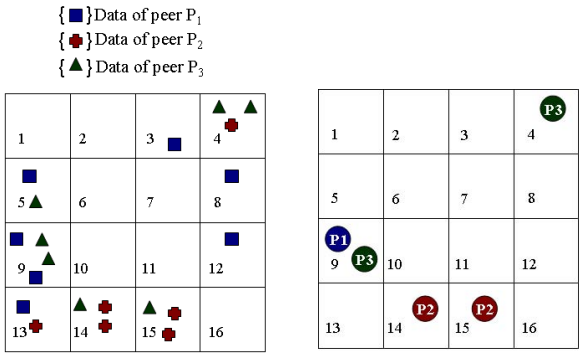
3. HYBRID OVERLAY NETWORK

The Hybrid Overlay Network (HON) organizes peers and data to perform an efficient similarity search based on range and nearest neighbor queries. The data contents of peers are represented by n-element feature vector, where each element is a particular feature or attribute associated with data object (e.g., color for an image, concept or key word for a text document). Since each data object is described by a feature vector, it can be seen as a point in a n-dimensional feature space.

The feature space is described by n features $f_1, f_2, ...f_n$. The basic idea is to define a partition of the feature space into cells and use the distribution of data objects over the cells as the basis for defining peer similarity, and computing query similarities to peers. Thus, two peers are considered similar if their contents are distributed on the same sub regions of the feature space. To define a partition of the feature space into cells, we evenly divide the range of values $[f_i^{min}, f_i^{max}]$ of a feature f_i into m_i intervals of size $\lceil \frac{f_i^{max} - f_i^{min}}{m_i} \rceil$, for $i = 1, 2...n$. We denote the resulting set of cells $\phi = \{\varphi_1, \varphi_2, ..., \varphi_m\}$, where m is given by $m = \prod_{i=1}^{n} m_i$.

3.1 Data and Peers Organization

Two steps are required to build HON. The first step consists in organizing data objects in the feature space. Each data object is described by a feature vector and corresponds to one point in the feature space. Therefore, it is mapped to one cell. Figure 1a shows the partition cells of a 2-



(a) Data organization (b) Peers organization

Figure 1: Hybrid Overlay Network

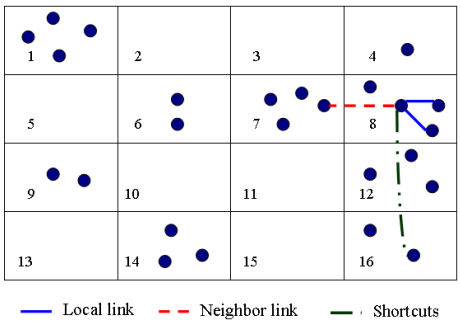


Figure 2: Indexing in HON

dimensional feature space using the features f_1 and f_2. Data of peers P_1, P_2 and P_3 are distributed over cells according to their description. For example, the data objects of P_2 are mapped to the cells 4, 13,14 and 15.

The second step consists in organizing peers in the feature space. Each peer is mapped to a set of cells containing its data objects. The mapping is done using a threshold value T. A peer is mapped to a cell only if it has a number of objects higher than T in the cell. Figure 1b shows that using a threshold $T = 1$, the peer P_2 is mapped only to the cells 14 and 15 because the number of its data objects in those cells is higher than 1.

3.2 Indexing and Searching

Each peer in HON maintains *local links* to its similar peers, i.e., peers within the same cells. Moreover, it maintains *neighboring links* to a random set of peers that belong to its adjacent cells. Recall that two cells are adjacent if they share $(d-1)$ hyperplane, where d is the dimensionality of the feature space. In addition, a peer maintains *shortcuts* as a separated layer, on top of local and neighboring links, to optimize the search. Shortcuts are created in the following manner. When a peer joins the system, it does not have any information about distant peers. Its first attempt to locate content using local and neighboring links. The lookup returns a set of peers that hold relevant answers. These peers are potential candidates to be inserted in the shortcut list. These shortcuts can be ranked using several criteria such as, utilization frequency, availability, etc. Figure 2 shows a peer belonging to cell 8 and maintaining local links to peers of the same cell and neighboring links to peers of adjacent cells 4, 7 and 12. In addition, it has a shortcut link to a peer in cell 16.

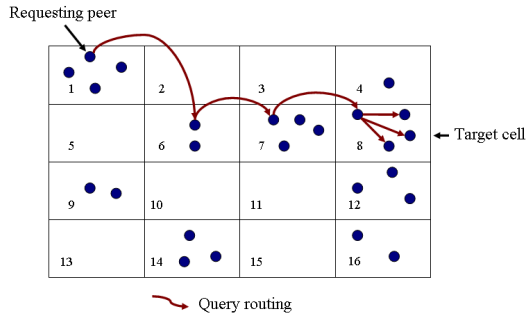
Local links are used to serve queries in local cells of the requesting peer. By contrast, neighboring links and shortcuts are used for routing queries from a group of similar peers to a distant group. Let us assume that a query is described by a set of values $(v_{Q1}, v_{Q2}, ...v_{Qn})$, where v_{Qi} is associated to the i^{th} feature. It corresponds to one point in the feature space. When a peer initiates the query Q, it first defines the cell where the query point falls, called target cell. A target cell is the cell that contains the relevant answers to the query Q. When the requesting peer defines the target cell, it forwards the query to one of its neighboring or shortcut peers that is the closest to the target cell. The query is forwarded form a peer to a neighboring peer till it reaches the target cell. When the query reaches the target cell it is flooded to all the contained peers to retrieve the relevant answers or similar ones. We note that a query can be a range query described by a set of range values $[Min_{Qi}, Max_{Qi}]$ for each feature i. In this case, more than one target cell might be selected. Thus, the query is processed for each target cell as described previously.

The size of cells, described by *cell granularity*, has a great impact on the search efficiency. The results presented in [6] show that the higher the cell granularity is the more accurate the similarity between data objects. On the other hand, a low cell granularity provides less precise and similar results. Moreover, the flooding within target cells might penalize the search performance if the cell has a high size and contains a high number of peers. Therefore, a large portion of irrelevant peers are involved in the query processing which increases the search cost. To limit the flooding overhead inside cells, we propose a caching mechanism that helps storing queries results for future use.

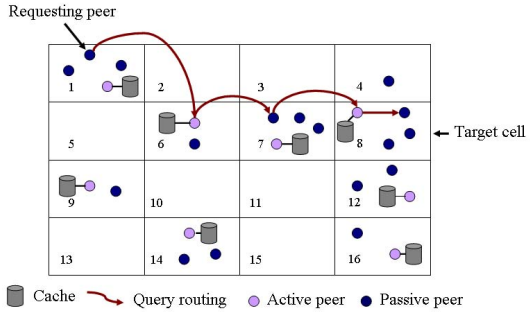
4. DISTRIBUTED CACHING SCHEMA

To improve search performance in HON, we apply a caching mechanism that reduces the flooding overhead inside cells. We place a cache in each non empty cell of the feature space. Therefore, the cache will keep trace of all the queries sent to the same cell. Queries sent to the same cell are similar and stored in the same cache. Unlike traditional caching assumptions, a cache in HON does not store the retrieved data objects. We assume that a cache stores the addresses of the peers that answered each query. In this manner, further queries generating a cache hit are not flooded to all peers in the cell but only to those that contain the relevant answers.

A cache is assigned to one peer in the cell called *active peer (AP)*. Peers with no associated cache are called *passive peers (PP)*. Each passive peer is registered with the active peer of its cell. When a given query is sent to the target cell, the relevant active peer checks its cache. If there is a cache Hit, the query is served from the peers whose addresses are stored in the cache. Otherwise, if there is a cache Miss, the query



(a) Flooding inside target cells



(b) Cache impact of flooding overhead

Figure 3: Caching in HON

is sent to all the peers contained in the target cell. Figure 3 shows the placement of the cache in the feature space. In, the first example illustrated in figure 3a, no cache is used, so the query is flooded to all the peers contained in the target cell. On the other hand, figure 3b shows the benefits of the cache that limits the flooding to a selected number of peers within the target cell.

The idea of placing the cache in each cell helps to collect all the similar queries in the same cache. Two queries that are mapped to the same cell are similar. Therefore, they share common target peers that are supposed to hold the relevant answers. In this way, similar queries are always sent to and stored in the same cache which helps increasing the success hit and avoiding redundancy.

A cache consists of a set of cached items, called *Query Segments*. They are used to record both queries and relevant peers that provide results to queries.

Definition 1 *(Query Segment)*
Given a query $Q = \{v_{Q1}, ..., v_{Qn}\}$ *defined over the set of features* $\{f_1, ..., f_n\}$. *A Query Segment S, is a tuple* $\langle S_Q, S_P \rangle$, *where* S_Q *is a submitted query and* $S_P = \{P_i\}_{i=1,m}$ *is the set of peers that provide results for* S_Q. *An empty query segment is defined by* $S{=}\langle \varnothing, \varnothing \rangle$.

A complete formal description of HON caching is given in our previous work [7].

5. CACHE MANAGEMENT

The granularity of cache management is at the query segment level. A query segment is stored and replaced as a whole. In this section, we first discuss query admission policy and then we present the cache consistency and replacement which are the two key components of cache management.

5.1 Cache Admission Policy

When the cache receives a query Q, it computes the similarity between Q and the queries stored in the cache using an Euclidian distance and a predefined similarity threshold. The result is the most similar query segment to the query Q from which is extracted a set of peers $\{P\}$ capable of answering the query Q. The requesting peer contacts all the peers in $\{P\}$ for results and selects the set of peers S_p that satisfy the query Q. The cache management in HON is based on two key points. First, on similarity that defines whenever a query can be served from the cache or not. Second, on the state of peer that can be online or offline. In the following, we give the definitions of cache miss and hit.

Definition 2 *(Cache hit)*
There is a cache hit if at least one query segment in the cache has a query that is similar to Q and if at least one peer of the returned peers is connected. A formal definition of the cache hit is given as follows: $\exists\ \langle S_Q, S_p \rangle \in C \mid S_Q$ *is similar to Q AND* $\exists\ P \in S_p \mid P$ *is connected.*

The similarity threshold for a cache hit is predefined. This threshold has to conserve the precision of answers to not decrease the search performance.

Definition 3 *(Cache miss)*
There is a cache miss if none of the query segment in the cache contains a query similar to Q, or if there are query segments in the cache with similar queries to Q but the corresponding peers are not connected. A formal definition of the cache miss is given as follows: $\forall\ \langle S_Q, S_p \rangle \in C$, S_Q *is not similar to Q OR* $\exists\ \langle S_Q, S_p \rangle \in C \mid S_Q$ *is similar to Q and* $\forall P \in S_p \mid P$ *is not connected.*

A challenging issue in HON caching is the use of similarity in defining the cache hit and miss. The similarity between two queries does not mean that their answers are stored in the same set of peers. Therefore, a cache hit might lead sometimes to irrelevant peers or partial set of peers that contains query answers. In this manner, we may loose a part of the information and generate partial results to queries which decrease the recall, i.e., the percentage of retrieved answers out of the existing answers in the whole network.

5.2 Cache Replacement Policy

In our work we investigate two replacement policies to study their impact on the search performance and particularly on the recall. As described previously, the main idea in P2P caching is to store the data in the cache for a short period of time. The first replacement technique we use in HON is LRU (Least Recently Used) which removes the data that has not been used for the longest period of time. The second policy is NFU (Not Frequently Used) which removes the less popular data in the cache.

We study in our work the impact of the replacement poli-

cies LRU and NFU in HON. Both LRU and NFU have advantages and disadvantages. Depending on the cache size, LRU helps to store the data in the cache for a short period of time. LRU makes a considerable difference with NFU, especially when new peers join the network providing new content. Since LRU removes the oldest data, it helps to refresh the content of the cache and take into account the new content that might join the network at any time. Therefore, the replacement policy LRU helps increasing the recall. By contrast, NFU policy keeps the most popular data on the cache. This helps to increase the success hit by providing answers to a large portion of user queries. However, NFU might ignore the network dynamicity and provide answers without taking into account new contents.

6. EVALUATION

We have shown in our previous work [6] that HON provides an efficient similarity search with a high success rate and recall. In this paper, we focus on studying the impact of caching on improving search performance and reducing the query scope.

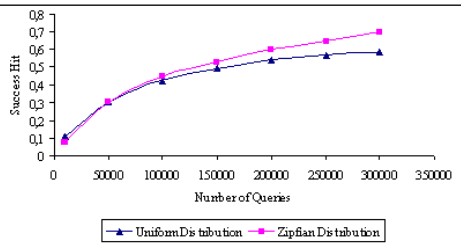
We have performed two main parts of simulation to evaluate caching performance. In the first part we focus on studying the success hit and the recall. In other words, how successful is the use of the cache to answer the queries, and how efficient it is in providing the maximum of the existing answers in the network. In the second part we have studied both replacement policies LRU and NFU and run some experiments to show their behavior.

6.1 Simulation Setup

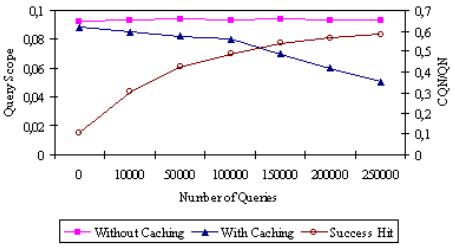
We run our simulation using 10,000 peers with 1,500,000 data objects. The average number of objects per peer is equal to 150. The feature space is divided into 10,000 cells. We initiate 3,000,000 queries over the network and we analyze the cache behavior. We have used the following metrics for the evaluation.

1. *Success Hit*: defined as the percentage of queries served from the cache. Let QN be the total number of queries, and CQN the number of queries served from the cache. The success hit is computed by QN/CQN.
2. *Query Scope*: is the fraction of peers in the system that is involved in the query processing. A smaller query scope increases system scalability. The query scope is computed by QP/N, where N is the total number of peers and QP is the average number of peers involved in the query processing.
3. *Recall*: represents the fraction of the relevant responses that has been retrieved. If a search retrieves only one hundred relevant responses out of three thousand available responses, then that search has a low recall. If it retrieves all the available responses to the query Q, it has a high recall. Let TR be the total number of the relevant responses for a query Q and RR be the number of the retrieved responses. The recall is computed by RR/TR.

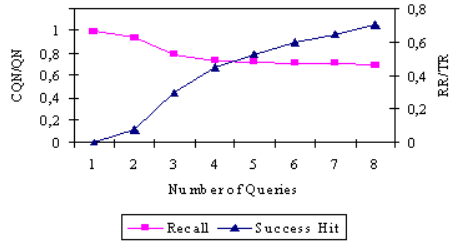
We have used two different query distribution. the first one is a *Uniform* distribution where each query has equal chance to be mapped to any cell of the feature space. The second one is a *Zipfian* distribution, where queries are mapped to few cells of the feature space. Approximately, 80% of queries are mapped (i.e, sent to and served from) 20% of cells.



(a) Impact of query distribution on success hit



(b) Impact of caching on the query scope



(c) Impact of caching on the recall

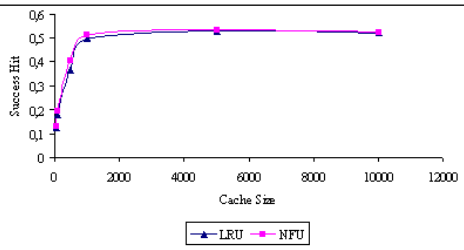
Figure 4: Caching impact on search efficiency

6.2 Data Distribution Impact on Success Hit

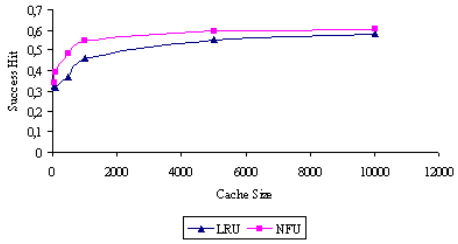
We analyze the behavior of the cache using different query distributions, and studying their impact on the success hit. Figure 4 shows the evolution of the success hit according to the the number of queries. At the begining, the cache is mostly empty, therefore the percentage of queries generating a cache hit is close to zero. This value increases when the cache starts storing queries' information. Figure 4a shows that a Zipfian distribution of queries increases the success hit. This is due to the fact that when using a uniform distribution, the queries are randomly chosen and they are rarely repeated, while a Zipfian distribution of queries provides 80% of similar queries increasing the success hit. Recall that a cache hit is based on the similarity between queries. Thus, when the number of similar queries increases, the success hit increases.

6.3 Caching Impact on Query Scope

We have analyzed the efficiency of the cache in reducing the flooding overhead inside cells. Thus, we have measured the impact of the cache on the query scope. Figure 4b shows the average query scope with and without using a cache. If we

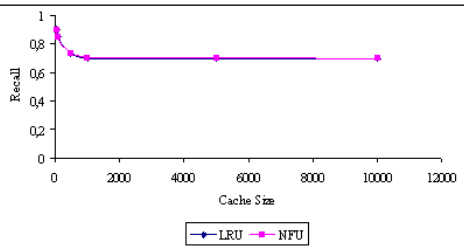


(a) Uniform query distribution

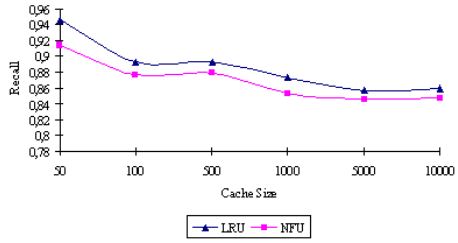


(b) Zipfian query distribution

Figure 5: Impact of replacement policies on success hit



(a) Uniform query distribution



(b) Zipfian query distribution

Figure 6: Impact of replacement policies on success hit

do not use a cache, we notice that the average query scope is equal to 10% of peers that are involved in processing queries. On the other hand, when we use a cache, we notice that the average query scope decreases when the query hit increases. In this experiment, the query scope decreases and reaches 5% of average query scope when the success hit is 60%. Here, we can notice that the cache decreases significantly the query scope which helps improving the search performance.

6.4 Caching Impact on Recall

To study the impact of the caching on the recall, we use the same experiments presented above. Figure 4c shows that the recall decreases when the success hit increases. At the beginning, when queries are not served from the cache, they are flooded to all the concerned peers and they retrieve all the relevant answers generating a recall equal to 100%. When the success hit increases, less precise answers are given from the cache, decreasing the recall. Figure 4c shows that the recall decreases and reaches 68% when the success hit reaches 70%.

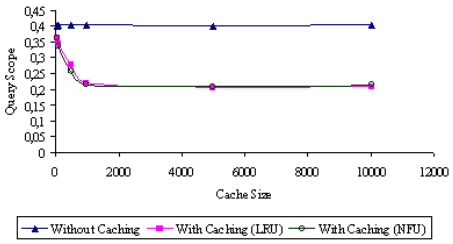
6.5 Replacement Policies

We have studied in our experiments the impact of two replacement policies on the success hit, the recall and the query scope. Figure 5 illustrates the impact of the replacement policies on the success hit using a uniform and a Zipfian query distribution. We notice that NFU policy helps increasing the success hit by keeping popular queries in the cache. The difference between LRU and NFU is significant when the query distribution is Zipfian as shown in figure 5b. Using a Zipfian distribution, few queries are initiated very often while many others are initiated rarely. Therefore, keeping queries in the cache based on their popularity increases the success hit comparing to the LRU policy. In the case of using a uniform distribution, no popular queries can be distinguished. Thus, we can hardly see the difference between LRU and NFU (cf. figure 5a).

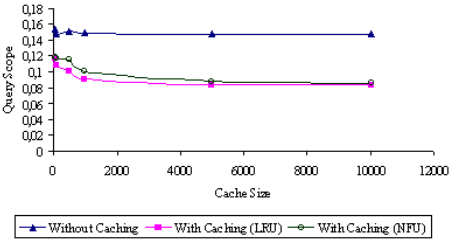
We have analyzed both replacement policies using different values of the cache size. Figure 5 shows that when the size of the cache is small ranging from 0 to 2000 segments, meaning that the replacement policies are actively used, we can notice clearly that NFU performs the success hit comparing to LRU. When the size increases, less replacement operations are processed, so the two replacement policies tend to provide more or less the same success hit.

We have studied the impact of the two replacement policies on the recall. Using LRU replacement policy and a small cache size, the queries are kept in the cache for a short period of time which is required in dynamic environments. Therefore, queries can take advantage of new content provided by new peers because the cache is refreshed frequently by removing old information. However, when using NFU policy, the popular queries results are kept in the cache for a long period of time. If a query is popular, it is always served from the cache even if more answers can be provided by some new peers. Figures 6b and 6a show that NFU policy decreases the recall comparing to LRU. The impact of the two replacement policies makes difference when the query distribution is Zipfian as shown in figure 6b, while using a uniform query distribution, both policies provide the same recall since they provide the same success hit as shown in figure 6a.

The last set of experiments of caching focuses on the impact of the replacement policies on limiting the flooding overhead by decreasing the query scope. As presented previously, the query scope is related to the success hit. When the success hit increases, the query scope deceases. This means that the flooding is limited when the query is served from the cache. We have shown above that NFU policy performs the success hit comparing to LRU. Therefore, the query scope is smaller using NFU than using LRU. Of course the impact of the replacement policies depends on the query distribution. We can notice in figures 7a and 7b that using a uniform distribution, the query scope provided by LRU and NFU is



(a) Uniform query distribution



(b) Zipfian query distribution

Figure 7: Impact of replacement policies on caching performance

almost the same while using a Zipfian distribution the query scope is decreased by NFU.

7. CONCLUSION

We have proposed in this paper a distributed caching schema to improve the performance of information retrieval in HON, a Hybrid Overlay Network for similarity search in a P2P system. We have presented a caching architecture that consists in assigning a cache to each non empty cell to collect similar queries. Thus, we increase the success hit and avoid redundancy. We have introduced new definitions for cache admission policies taking into account the dynamic behavior of P2P networks. Moreover, we have investigated two replacement policies namely, LRU and NFU.

To study the performance of the query processing and caching, we have run a set of experiments studying different metrics such as query recall, query scope and success hit. We have shown through these experiments that caching decreases significantly the query scope allowing an optimal search. The experiments have shown that LRU replacement policy helps increasing the recall by keeping data for a short period of time in the cache. On the other hand, NFU policy increases the success hit by providing answers to a large portion of user queries. However, it might ignore the network change and provide answers without taking into account new contents.

8. REFERENCES

[1] L. A. Adamic, R. Lukose, A. Puniyani, and B. Huberman, "Search in power law networks," vol. 64, no. E, pp. 46 135–46 143, 2001.

[2] A. Chankhunthod, P. Danzig, C. Neerdaels, M. Schwartz, and K. Worrell, "Hierarchical internet object cache," *In Proceedings of the USENIX Technical Conference*, 1996.

[3] E. Cohen and S. Shenker, "Replication strategies in unstructured peer-to-peer networks," *In Proceedings of the 2002 conference on Applications, Technologies, Architectures and protocols for computer communications*, pp. 177–190, 2002.

[4] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[5] Gnutella, "http://www.gnutella.com," 2003.

[6] M. Kacimi and K. Yetongnon, "Density-based clustering for similarity search in a p2p network," *In proceedings of the 6th IEEE Symposium on Cluster Computing and the Grid*, 2006.

[7] M. Kacimi, K. Yetongnon, Y. Ma, and R. Chbeir, "Distributed caching in a cluster-based hybrid overlay network for p2p systems," *In the proceedings of the 18th International Conference on Parallel and Distributed Computing Systems*, 2005.

[8] KAZZA, "http://www.kazaa.com/," 2002.

[9] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A scalable and dynamic emulation of the butterfly," *In Proceedings of the 21st annual ACM symposium*, pp. 183–192, 2002.

[10] E. P. Markatos, "Tracing a large-scale peer to peer system: An hour in the life of gnutella," *In Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, p. 65, 2002.

[11] S. Patro and Y. C. Hu, "Transparent query caching in peer-to-peer overlay networks," *International Parallel and Distributed Processing Symposium (IPDPD'03)*, no. 32, 2003.

[12] S. Paul and Z. Fei, "Distributed caching with centralized control," *Computer Communications journal*, vol. 24, no. 2, pp. 256–268, 2001.

[13] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," *In Proceedings of ACM SIGCOMM*, 2001.

[14] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," Carnegie Mellon University, Tech. Rep., Jan 2004.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," *ACM SIGCOMM*, pp. 149–160, 2001.

[16] C. Wang, L. Xiao, Y. Liu, and P. Zheng, "Distributed caching and adaptive search in multilayer p2p networks," *In Proceedings of the 24th International Conference on Distributed Computing Systems (IDCS'04)*, 2004.

[17] S. Wang, L. Xiao, Y. Liu, and P. Zheng, "Dicas: An efficient distributed caching mechanism for p2p systems," *IEEE Transaction on Parallel and Distributed Systems*, 2006.

[18] A. Wierzbicki, N.Leibowitz, M.Ripeanu, and R. Wozniak, "Cache replacement policies revisited: The case of p2p traffic," *In Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*, 2004.