A Query-Adaptive Partial Distributed Hash Table for Peer-to-Peer Systems

Fabius Klemm, Anwitaman Datta, Karl Aberer

School of Computer and Communication Sciences
Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland
{Fabius.Klemm, Anwitaman.Datta, Karl.Aberer}@epfl.ch

Abstract. The two main approaches to find data in *peer-to-peer* (P2P) systems are *unstructured* networks using flooding and *structured* networks using a distributed index. A distributed index is usually built over *all keys* that are stored in the network whether they are queried or not. Indexing all keys is no longer feasible when indexing *metadata*, as the key space becomes very large. Here we need a query-adaptive approach that indexes only keys worth indexing, i.e. keys that are queried at least with a certain frequency. In this paper we study the cost of indexing and propose a query-adaptive *partial distributed hash table* (PDHT) that does not keep all keys in the index. We model and analyze a scenario to show that query-adaptive partial indexing outperforms pure flooding and "index-everything" strategies. Furthermore, our scheme is able to automatically adjust the index to changing query frequencies and distributions.

Keywords: peer-to-peer (P2P), partial distributed hash table (PDHT), query-adaptive indexing, metadata.

1 Introduction

There have been several proposals to store and retrieve data in decentralized unreliable peer-to-peer networks. In most of the solutions the two alternatives so far have been to index *all or nothing*. In unstructured networks, such as Gnutella, peers use flooding or multiple random walks [ChRa03, LvCa02] to resolve queries and do not build and maintain any index. These mechanisms can be used for arbitrary, complex search requests on *metadata* as they are not restricted to certain keys to find values in the network. On the other hand queries generate a large number of messages.
In structured peer-to-peer networks [Aber01, RaFr01, RoDr01, StMo01], also called distributed hash tables (DHTs), peers collaborate to construct and maintain a distributed index, which allows very efficient searches, but are, however, restricted to searches on the indexed keys [HaHe02]. Moreover, traditional DHTs do not consider the query distribution and devote equal resources to all keys. Such drawbacks of DHTs are discussed in [ChRa03].

When it comes to indexing metadata, a big difficulty lies in selecting *useful* keys for the index. Which metadata is actually used in queries depends on the application. Furthermore, the popularity of keys can change dramatically over time. Let us consider a distributed, decentralized peer-to-peer news system. Peers generate *news articles*, which are described by metadata. These metadata files consist of *element-value* pairs, such as title = "Weather Iráklion", author = "Crete Weather Service", date = "2004/03/14", and size = "2405". Queries may contain predicates on the different metadata attributes, such as *element1* = *value1* AND *element2* = *value2*. In case we decide to index a specific metadata attribute we generate keys by hashing single or concatenated key-value pairs, such as proposed in [FeBi04]. From this little example we can already see that indexing *key1* = *hash*(title = "Weather Iráklion" AND date = "2004/03/14") makes much more sense than indexing *key2* = *hash*(size = "2405") as *key1* is much more likely to be queried.

In this paper we propose an algorithm that selects the keys that are *worth* indexing (such as *key1*). We first study the cost of maintaining a key-value pair in the index in the presence of peers going on-/off-line and frequent key updates. Routing table maintenance costs are considerable, as P2P clients are extremely transient in nature [ChRa03]. Searching a key in the index is much cheaper than searching an unstructured network. However, we claim that the cost of keeping a rarely queried key (such as *key2*) in the index can become higher than the cost saving offered by efficient index search. We therefore introduce an analytical model, to decide whether a key with a given query frequency is worth indexing. Second, we propose an extension for DHTs, which does not index all keys occurring in the network but is able to select those keys that are worth indexing. Our partial indexing approach also adjusts to changing query distributions, which are typical of contemporary P2P applications, such as file sharing.

Note that our proposal is generic enough such that it can be used for any of the DHT based systems. The main contribution of this paper is to analyze the cost of indexing, which allows to decide whether a key is worth indexing and thus to realize a hybrid peer-to-peer system based on a query adaptive partial DHT. The main objective of the analysis has been to provide an intuitive understanding of such hybrid designs of P2P systems, and we make several simplifying assumptions to obtain a qualitative understanding. Given the diversity of existing DHT system designs, a generic analysis for all DHT based systems is beyond the scope of this work. Therefore, we concentrate on what can be called the traditional DHTs [Aber01, RaFr01, RoDr01, StMo01]. However, the analysis is generic enough such that it can be adapted to suit most other DHT proposals. We point out such simplifying assumptions wherever applicable.

The paper is structured as follows: In Sections 2 and 3 we shall present an analytical model for partial indexing. Section 4 examines a realistic example of partial indexing. In section 5 we propose a decentralized selection algorithm that dynamically chooses keys worth indexing and purges unnecessary keys from the index. Section 6 finishes with discussion and conclusions.

2 To Index or Not to Index?

In this section we propose an analytical model to decide whether a key is worth indexing. Keys that are not queried frequently enough are not worth indexing as they only increase the size of the index. The bigger the index, the more peers are necessary to store the index. On the other hand, if there are no queries, there is no need to maintain a DHT.
Depending on the *query distribution*, each key is queried with a certain frequency. We are now interested in finding the lowest query frequency *fMin* a key must have to be worthwhile indexing.
The decision to index a key depends on the following variables:

- *cIndKey*: The cost of storing one key in the index for one round[1].
- *cSUnstr*: The cost of searching a key in an unstructured network.
- *cSIndx*: The cost of searching the index.
- $fQry_k$: The number of queries for key *k* per round (*query frequency*).

A key *k* should be indexed if it is queried frequently enough to amortize the indexing cost:

$$fQry_k \cdot (cSUnstr - cSIndx) - cIndKey > 0 \quad (1)$$

If a key is not queried frequently enough, it is better not to index it. Since $cSUnstr - cSIndx > 0$ the minimum frequency a key must have to be worthwhile indexing is:

$$fQry_k > \frac{cIndKey}{cSUnstr - cSIndx} \quad (2)$$

Therefore we set *fMin* to the smallest value $fQry_k$ such that (2) holds.
We now assume that queries for keys are *Zipf distributed* with parameter α [Srip01] and that there are *keys* number of unique keys. The probability of a query for the key at position *rank* is therefore:

$$prob_{rank} = \frac{rank^{-\alpha}}{\sum_{x=1}^{keys} x^{-\alpha}} \quad (3)$$

With *numPeers* peers and an average query frequency of *fQry* per peer per round, all peers together send a total of $numPeers \cdot fQry$ queries each round. A key therefore has the following probability of being queried at least once per round:

$$probT_{rank} = 1 - (1 - prob_{rank})^{numPeers \cdot fQry} \quad (4)$$

With (4) we now set *maxRank* to the highest *rank* such that $probT_{rank} \geq fMin$, i.e. *maxRank* is the number of keys worth indexing. The probability that a random Zipf distributed query can be answered from the index is:

[1] One round is a fixed period of time. We shall later set one round to one second.

$$pIndxd = \frac{\sum_{x=1}^{maxRank} x^{-\alpha}}{\sum_{x=1}^{keys} x^{-\alpha}} \quad (5)$$

3 Model

As we have seen in the preceding section, the decision to index depends on the cost of indexing a key (*cIndKey*), the cost of searching the unstructured network (*cSUnstr*), and the cost of searching the index (*cSIndx*). We are now looking for realistic models for these parameters assuming standard solutions for P2P overlay networks being used. As is a standard practice in P2P systems we consider the number of messages as the main cost (as opposed to storage and processing cost).

3.1 cSUnstr

We assume that the unstructured network has a Gnutella-like topology, where each peer has a few open connections to other peers. However, the Gnutella flooding-based query algorithm is not optimal even for unstructured networks. We therefore assume that a search algorithm is used that consumes less network traffic, such as multiple random walks as presented in [LvCa02]. We will explain in the next section that we replicate keys with a certain factor at random peers. We furthermore assume that the search algorithm in the unstructured network finds any key if it exists in the network. When searching an unstructured network, some peers receive several copies of the same query, depending on the network connectivity [LvCa02]. We therefore use a message duplication factor *dup*. With *numPeers* and a random replication with factor *repl* the cost of searching an unstructured network is then:

$$cSUnstr = \frac{numPeers}{repl} \cdot dup \ \text{[msg]} \quad (6)$$

3.2 cSIndx

Search in traditional DHTs is with logarithmic cost[2]. With *maxRank* keys in the index, a given replication factor, and each peer having a storage capacity of *stor* keys, we need *numActivePeers* peers to store the index. We assume that if the total number of

[2] Recently, several DHTs with sub-logarithmic search costs have been proposed, such as in [Man04]. However, in this paper we concentrate on traditional DHTs [Aber01, RaFr01, RoDr01, StMo01] as both the qualitative insights and the proposed algorithm will hold even though the quantitative results will change.

peers is greater than the number of peers necessary to store the index (*numPeers* > *numActivePeers*), only *numActivePeers* peers participate in building and maintaining a DHT. For the remaining peers, to perform searches, it is sufficient to know at least one online peer that is participating in the DHT (i.e. the set of *numActivePeers*). The cost of searching the index in a binary key space[3] is then:

$$cSIndx = \frac{1}{2} \cdot Log_2(numActivePeers) \text{ [msg]} \quad (7)$$

3.3 cIndKey

The cost of keeping a key in the index for one round depends on the following two factors:

3.3.1 Routing table maintenance cost: cRtn

Peers continuously join and leave the system. To assure a certain level of routing reliability, the peers must keep their routing tables up-to-date. One possible strategy is to *probe* routing entries with a given rate to detect offline peers [MaCa03]. The challenge thereby is to adapt the probe rate to the current network situation, for example by estimating the network size and the peer's online characteristics. The amount of probe messages depends on the routing table size, which is $O(Log(numActivePeers))$. Stale routing entries can be replaced with low overhead by piggybacking routing information on queries. Therefore, we need only messages to *detect* stale routing entries (by probing) but assume no additional messages to repair those routing entries. The amount of probe messages is an application dependent environment constant (such as determined in [MaCa03]), which we call *env*.

Suppose that we need *numActivePeers* peers to build a DHT big enough to index *maxRank* keys. The maintenance cost for routing tables *per key* per second is then: The cost to detect stale routing entries (*env*) multiplied by the size of the routing table $Log_2(numActivePeers)$ times the number of participating peers (*numActivePeers*) divided by the number of keys kept in the index (*maxRank*):

$$cRtn = env \cdot Log_2(numActivePeers) \cdot numActivePeers / maxRank \text{ [msg/s]} \quad (8)$$

3.3.2 Update cost: cUpd

The second part of *cIndKey* is the cost of inserting, overwriting, or deleting a key in the index thereby assuring consistency among the replicas. An update works as follows: The replicas in the index maintain an *unstructured replica subnetwork* among each other. When updating a key, it is inserted at one responsible peer in the index at the cost of searching the index (*cSIndx*) and then gossiped to the other responsible peers in the subnetwork of replicas. Therefore, the update cost also depends on the replication factor *repl*, which is given by the application.

[3] For simplicity we assume a binary key space. However, the analysis can also be generalized for a *k*-ary key space.

[DaHa03] studied the cost of updates between replicas based on a hybrid push/pull rumor spreading algorithm. Peers that are offline and go online again pull for missed updates. We assume a message duplication factor of *dup2* for flooding the replica subnetwork. Given that *fUpd* is the average update frequency per key per second the cost of updating a key is:

$$cUpd = (cSIndx + repl \cdot dup2) \cdot fUpd \text{ [msg/s]} \quad (\mathbf{9})$$

Thus we obtain the cost of indexing a key per round:

$$cIndKey = cRtn + cUpd \text{ [msg/s]} \quad (\mathbf{10})$$

The cost of searching the unstructured network (*cSUnstr*) is usually considerably higher than the cost of searching the index (*cSIndx*) and the cost of keeping a key in the index (*cIndKey*). However, we will see that searching rarely queried keys in the unstructured network is cheaper than proactively keeping them in the index. Therefore, only keys that are queried at least with a certain frequency *fMin* should stay in the index.

4 Evaluation of the model

We now provide a simple scenario in order to instantiate our model with concrete parameters. In doing so, we choose values as have been observed in the context of P2P networks by various researchers [LvCa02, MaCa03, Srip01]. Thus, we endeavor to provide a simple and practical decentralized solution under realistic assumptions for a longstanding drawback of DHTs, that of judiciously indexing metadata.

We imagine a news system with the following characteristics: It should be able to store 2,000 unique news articles, randomly replicated with a certain factor. We assume that there exists a mechanism to determine a proper replication factor for the index and content files (news articles) to meet target levels of availability and to avoid unnecessary high update cost [VaCh02]. Such mechanisms lie beyond this work and are therefore not further discussed. Index and content are replicated with the same factor to assure the same search reliability in structured and unstructured networks. In our analysis we use a replication factor of 50.

As discussed in Section 1, for each article we generate 20 keys from the metadata describing the article. It is a standard approach in information retrieval to avoid indexing stop words, such as “the”, “and”, etc. We assume that the set of such stop words is globally known to all peers in the system and are ignored. To index 2,000 news articles we therefore get 40,000 keys. Each peer has a storage capacity of 5 articles plus a cache of 100 *key-value pairs* that can be used for indexing. With replication factor of 50 we therefore need 20,000 peers to store and index all articles.

Each article is replaced every 24 hours on average. New articles are actively replicated together with their metadata files. The average *query frequency per peer* varies from one query every 30 seconds, in very busy periods of the day, to one every 2 hours, in calmer times. Thus, with 20,000 peers and 40,000 keys, the average *key query/update ratio* varies between 1440/1 and 6/1. Furthermore, the queries are Zipf-distributed with *alpha* = 1.2 as observed in [Srip01].

For this analysis we use the route maintenance cost that [MaCa03] studied for Pastry. Using a Gnutella trace with 17,000 peers, they analyzed that around 1 message per peer per second is necessary. With (8) we therefore get a routing maintenance constant of $env = 1/Log_2(17{,}000) \approx 1/14$. This constant might be different in other maintenance approaches and environments. In this scenario, the maintenance cost (*cRtn*) clearly outweighs the update cost (*cUpd*).
We first assume that each peer knows which keys it can find in the index and for which it has to do a broadcast search[4]. This assumption is not realistic but at the moment we are interested in the best performance possible with partial indexing (lower bound in terms of messages in the given system model). We will discuss in the next section a more realistic environment, in which peers do *not* know whether a key is in the index. The following table summarizes the parameters:

Table 1. Parameters of the sample scenario.

Description	Param.	Value
Total number of peers	*numPeers*	20,000
Number of peers building the DHT	*numActivePeers*	
Number of unique keys	*keys*	40,000
Storage capacity for indexing per peer	*stor*	100
Replication factor	*repl*	50
α of query Zipf distribution	α	1.2 [Srip01]
Frequency of queries per peer per second	*fQry*	1/30 1/s to 1/7200 1/s
Avg. update freq. per key	*fUpd*	$1/(3600 \cdot 24)$ 1/s
Route maintenance constant	*env*	1/14 [MaCa03]
Message duplication factors	*dup* *dup2*	1.8 [LvCa02] 1.8

We can now calculate the total cost for pure indexing and broadcast searches as well as for partial indexing:
Total cost of indexing all keys: Cost of the full index per second, where *maxRank* = *keys*, plus the cost of searching the index (*cSIndx*). The total number of queries per second is $fQry \cdot numPeers$:

$$indexAll = keys \cdot cIndKey + fQry \cdot numPeers \cdot cSIndx \text{ [msg/s]} \quad (11)$$

Total cost of searching all queries in the unstructured network:

$$noIndex = fQry \cdot numPeers \cdot cSUnstr \text{ [msg/s]} \quad (12)$$

[4] We use "broadcast search" and "search in the unstructured network" interchangeably.

Total cost for ideal partial indexing:
We index only the *maxRank* most popular keys, which costs *cIndKey* per key per second. If a query can be answered from the index (with probability *pIndxd*), we have only the index search cost (*cSIndx*). Otherwise (1 - *pIndxd*) we have to search the unstructured network (*cSUnstr*).

$$partial = maxRank \cdot cIndKey + pIndxd \cdot fQry \cdot numPeers \cdot cSIndx + \quad (1 - pIndxd) \cdot fQry \cdot numPeers \cdot cSUnstr \text{ [msg/s]} \tag{13}$$

We used the analytical model to evaluate the behavior of the system for a changing query load (*fQry*). The following figures show the results.

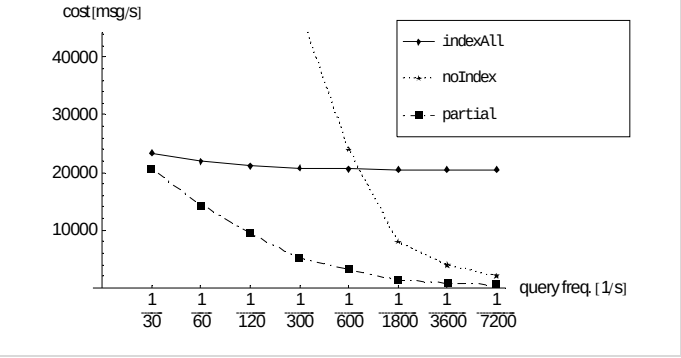

Fig. 1. Query frequency per peer (*x*-axis) vs. total sent messages per second (*y*-axis) when all keys are indexed (*indexAll*, solid), when all queries are answered by broadcast (*noIndex*, dashed stars), and for ideal partial indexing (*partial*, dashed squares).

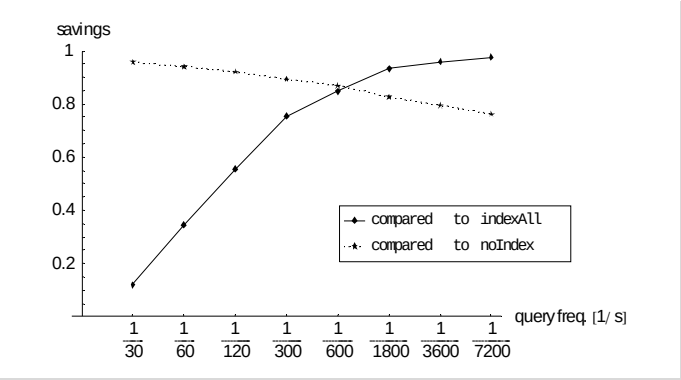

Fig. 2. Savings of ideal partial indexing compared to indexing all keys (*indexAll*, solid) and compared to broadcasting all queries (*noIndex*, dashed).

Fig. 1 shows the cost when indexing and broadcasting all queries, and with ideal partial indexing. Ideal partial indexing is considerably cheaper for all query frequencies as the savings in Fig. 2 show. In Fig. 3 we can see that the index size decreases with

lower query frequencies as the index only stores the keys worth indexing. As the queries are Zipf distributed even a small index can answer a high percentage of queries.

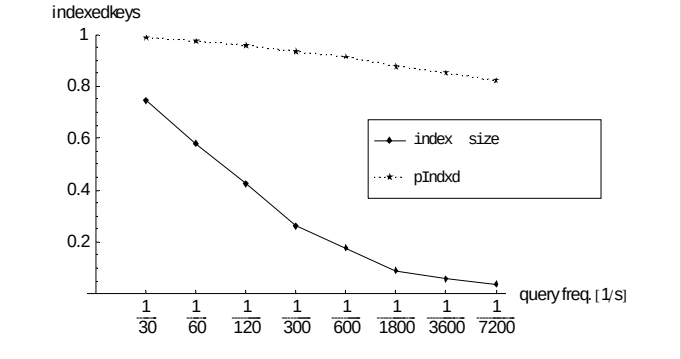

Fig. 3. Percentage of indexed keys with ideal partial indexing (*index size*, solid) and percentage of queries that can be answered from the index (*pIndxd*, dashed).

5 Selection algorithm

In the preceding section we made the idealizing assumption that every peer knows whether a key should be indexed. In this section we now present a simple decentralized algorithm to select keys worth indexing.

5.1 How to select worthwhile keys?

When a peer wants to answer a query, it first searches the index. If there is no result, the peer initiates a broadcast search and inserts the resulting key-value pair into the index. Each key has an expiration time *keyTtl*, which determines how long the key stays in the index. The expiration time of a key is reset to a predefined value whenever the peer that stores the key receives a query for it. Therefore, peers evict those keys from their local storage that have not been queried for *keyTtl* rounds. This mechanism has the effect that only frequently queried keys stay in the index whereas the unpopular keys, which are not worth indexing, time out. This approach does not take the relative frequency of queries into account, but only the temporal Boolean distribution of whether there was any query for a particular key.
With (3) and (4), the probability that a query can be answered from the index, i.e. that the key has been queried at least once in the last *keyTtl* rounds, is:

$$pIndxd = \sum_{rank=1}^{keys} \left(\left(1 - \left(1 - probT_{rank} \right)^{keyTtl} \right) \cdot prob_{rank} \right) \quad (\mathbf{14})$$

The number of keys in the index is:

$$keys = \sum_{rank=1}^{keys} \left(1 - \left(1 - probT_{rank}\right)^{keyTtl}\right) \quad (\mathbf{15})$$

Purging timed-out keys leads to poor replication synchronization. To improve the query success rate, peers propagate queries in the unstructured replica subnetwork if they cannot answer them. The index search cost therefore increases by the cost of flooding the replica network:

$$cSIndx2 = cSIndx + repl \cdot dup2 \text{ [msg]} \quad (\mathbf{16})$$

A peer first searches the index (*cSIndx2*). If it does not find the key in the index, it searches the unstructured network (*cSUnstr*) and inserts the resulting key into the index (*cSIndx2*). Therefore, proactive updates (*cUpd*) are no longer necessary and the cost of keeping a key in the index consists only of the routing cost (*cRtn*). The cost of partial indexing therefore is:

$$partial = keys \cdot cRtn + pIndxd \cdot fQry \cdot numPeers \cdot cSIndx2 + \quad (\mathbf{17})$$

$$(1\text{-}pIndxd) \cdot fQry \cdot numPeers \cdot (cSIndx2 + cSUnstr + cSIndx2) \text{ [msg/s]}$$

Fig. 4 shows the savings with the proposed selection algorithm. We see that the algorithm causes some overhead, but partial indexing still realizes substantial savings, in particular for average query frequencies.

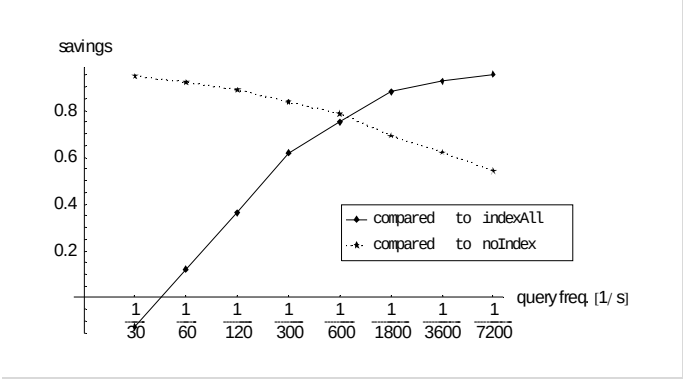

Fig. 4. Savings with proposed selection algorithm compared to indexing all keys (*indexAll*, solid) and compared to broadcasting all queries (*noIndex*, dashed).

With the selection algorithm the cost of indexing is higher than with ideal partial indexing (which results in lower savings) for four reasons: I. Not all keys that are supposed to be in the index are in the index at the time they are queried. We chose a *keyTtl* of 1/*fMin*. A key worth indexing can time out before it is queried again. II. Keys that are not worth indexing are inserted into the index for *keyTtl* rounds. III. Index search cost (*cSIndx2*) is higher than ideal search cost (*cSIndx*). IV. A peer does not know whether a query is indexed. It therefore always searches the index and then

also broadcasts the query if necessary. Nevertheless, there are still considerable savings compared to strategies that index all keys or broadcast all queries (except for very high query frequencies) as shown in Fig. 4.

5.1.1 Choosing *keyTtl*

It is important that peers insert keys into the index with the right expiration time (*keyTtl*). The value of *keyTtl* can be calculated by estimating *cSUnstr*, *cSIndx*, and *cIndKey*. A too small value results in fewer savings at high query frequencies, a too big value at lower frequencies. Analytical results show that an estimation error of ±50% of the ideal *keyTtl* decreases the savings only slightly. A mechanism to self-tune *keyTtl* based on the query distribution and frequency is part of future work.

5.2 Implementation

We have been implementing a simulator for partial indexing with P-Grid [Aber01]. Preliminary simulation results show that the selection algorithm works well and that P-Grid adapts to changing query distributions. We are currently implementing extensions to simulate efficient routing table maintenance and algorithms that assure a certain replication depending on the online characteristics of peers. Once stable, these algorithms will be integrated in the P-Grid P2P system, which has been implemented in Java (www.p-grid.org).

6 Discussion & Conclusions

We have argued that in structured P2P systems the indexing cost cannot be neglected, particularly the cost of maintaining routing tables. Depending on the query frequency of a key, it can be cheaper not to index it. To outline a design for a query adaptive partial DHT, we first provided an analysis based on global knowledge. Then we proposed a very simple, nonetheless effective mechanism for selective (partial) indexing based on only locally available information at peers, without the need of global coordination. Our self-organizing and adaptive mechanism does not make the system theoretically optimal as we do not assume global coordination. However, the results show that our approach leads to performance benefits and compares well with the theoretical optimal solution. It adapts to changing query frequencies and distributions, which is especially useful when indexing metadata, as the range of the key space that is actually queried depends on the applications and can dramatically change over time. Future work includes refinements of the analytical model and improvements in the proposed selection algorithm and its implementation.

References

[Aber01] K. Aberer. *P-Grid: A Self-Organizing Access Structure for P2P Information Systems.* Sixth International Conference on Cooperative Information Systems (CoopIS 2001), Trento, Italy 2001

[ChRa03] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker: *Making Gnutella-like P2P Systems Scalable.* Proceedings of ACM SIGCOMM 2003

[DaHa03] A. Datta, M. Hauswirth, and K. Aberer. *Updates in Highly Unreliable, Replicated Peer-to-Peer Systems.* 23rd International Conference on Distributed Computing Systems (ICDCS), 2003.

[FeBi04] P. Felber, E. Biersack, L. Garces-Erce, K. W. Ross, G. Urvoy-Keller. *Data Indexing and Querying in P2P DHT Networks*. ICDCS 2004, Tokyo, Japan

[HaHe02] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. *Complex queries in dht-based peer-to-peer networks.* In Proc. of IPTPS '02, 2002.

[LvCa02] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. *Search and replication in unstructured peer-to-peer networks.* In 16th international conference on Supercomputing, June 2002.

[MaCa03] R. Mahajan, M. Castro, and A. Rowstron. *Controlling the cost of reliability in peer-to-peer overlays.* In IPTPS, 2003.

[Man04] G. S. Manku. *The power of lookahead in small-world routing networks.* In STOC, 2004.

[RaFr01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network.* In SIGCOMM, Aug. 2001.

[RoDr01] A. Rowstron and P. Druschel. *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.* In IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), November 2001.

[Srip01] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. In O'Reilly's, www.openp2p.com, 2001.

[StMo01] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.* In Proceedings of ACM SIGCOMM 2001.

[VaCh02] A. Vahdat, J. Chase, R. Braynard, D. Kostic, P. Reynolds, A. Rodriguez. *Self-Organizing Subsets: From Each According to His Abilities, To Each According to His Needs.* First International Workshop on P2P Systems, March 2002