Scalable, Efficient Range Queries for Grid Information Services

Artur Andrzejak and Zhichen Xu
Hewlett-Packard Laboratories, Palo Alto
{artur_andrzejak,zhichen_xu}@hp.com

Abstract

Recent Peer-to-Peer (P2P) systems such as Tapestry, Chord or CAN act primarily as a Distributed Hash Table (DHT). A DHT is a data structure for distributed storing of pairs (key, data) *which allows fast locating of* data *when a* key *is given.*

To facilitate efficient queries on a range *of* keys*, we propose a CAN-based extension of this DHT-functionality. The design of our extension suggests several range query strategies; their efficiency is investigated in the paper. A further goal is to enhance the routing aspects of current DHT-systems so that also frequently changing data can be handled efficiently. We show that some relatively simple approaches are able to reduce the communication overhead in this case.*

The design of the system is driven by its application as a part of the information infrastructure for computational grids. Such grids provide an infrastructure for sharing computing resources; an information infrastructure is their inherent part which collects resource data and provides search functionality. Our approach complements current solutions such as MDS-2 by adding self-organization, fault-tolerance and an ability to efficiently handle dynamic attributes, such as server processing capacity.

We evaluate our system in this context via a simulation and show that its design along with particular query and update strategies meet the goals of scalability, communication-efficiency and availability.

1. Introduction

Computational grids provide means for sharing computing power, storage capacity, instruments, data, and applications. They involve many thousands or millions of distributed-computing devices, linked together in local or regional networks, which in turn are interconnected on a global level. In addition to providing computing power for solving the grand–challenge problems, grid is envisioned as a technique to remove one of the major pitfalls of today's information technology systems, namely capacity planning [5].

Unlike traditional computing systems where all the resources in the system are under the direct control of a single organization, grid resources are geographically distributed, heterogeneous in nature, owned by different individuals or organizations with possibly different policies [11].

To enable efficient and appropriate uses of the resources from both the systems' and applications' perspectives, it is important to provide means to keep track of the availability and attributes of millions of resources. In grid this functionality is provided by an *information service.* The most widely known implementation of such a service is MDS-2 [3].

In an information service computing resources are characterized by sets of attributes, as for example the type of the operating system, network address, CPU speed or storage capacity. A fundamental function of such a system is the search for resources with specific combinations of attribute values. Due to a large number of resources, indexing of the attributes becomes necessary.

This problem can be attacked by a P2P-based, distributed indexing infrastructure. The advantages of such an approach are the self-organizing characteristics, fault-tolerance and scalability inherent to P2P-systems. We believe that these properties make the P2P-based information services attractive enough to complement or partially substitute current (non-P2P) solutions.

While all P2P systems provide distributed storage and search functionality, they differ mostly in the degree of search efficiency and in success guarantees. In earlier systems such as Gnutella [6] and Freenet [2], a search request "floods" a limited (overlay network) neighborhood of the requester. This approach cannot provide any guarantee whether a search finds the desired data even if this data is present. Its second disadvantage is the large amount of traffic generated. More recent systems such as Tapestry [8], Chord [11] or CAN [9] feature a deterministic structure by mimicking a DHT mentioned in the abstract. As a result, they are able to locate each data present in the system; those systems also provide performance guarantees in terms of logical hops while taking advantage of network proximity.

Different types of attributes managed in an information infrastructure might require different indexing mechanisms. Attributes which have a limited number of values (e.g. the type of the operating system) can be handled efficiently by the above-mentioned systems implementing a DHT. On the other hand, for "continuous" types of attributes, such as processing or storage capacity, querying of attribute *ranges* is useful: first, finding resources with an *approximate* attribute value might be sufficient; furthermore, we might want to locate all resources with attribute values larger (or smaller) than a certain number.

Unfortunately, the systems implementing a DHT do not support queries of ranges. In their current versions, each discrete value in a range must be queried individually, which is infeasible in the most cases. An orthogonal aspect where those systems perform poorly is the efficiency of handling fast-changing attributes. Certain resource characteristics, such as processing capacity, might change dramatically in a short time period. This implies short update intervals and as a consequence increased network traffic. None of the above-mentioned systems provide mechanisms to counter this problem.

Driven by these limitations, we extend the CAN-based DHT-system into an indexing infrastructure which allows querying of ranges and supports efficient handling of dynamic data. Furthermore, we take advantage of our prior work, expressway routing [13], to further cut down costs of searching and updating. Our work provides foundation for a self-organizing and scalable implementation of such parts of a grid information infrastructure as Grid Index Information Service (GIIS) [3], which provides a coherent image of distributed grid resources and allows searching for specific resources.

Our contributions are the following:

- Extension of CAN for efficient handling of range queries by using Space Filling Curves as hash functions.
- Design and evaluation of range query strategies which are simple yet efficient in terms of network traffic and time.
- Design and comparison of attribute value update strategies which reduce the overhead caused by a P2P-infrastructure.
- A detailed simulation study based on synthetic data and real data center traces.

The remainder of the paper is organized as follows: Section 2 discusses related work. Section 3 describes distributed indexing infrastructure for range queries. Section 4 presents techniques to perform efficient query and update. Section 5 evaluates our approaches using simulation. Section 6 concludes and sketches the future work.

2. Related Work

There are quite a few existing projects that address the problem of search in a grid information service.

MDS-2 provides directory services for grids [3]. It uses LDAP as a uniform interface for accessing and managing static and dynamic information about the status of a grid and its components. It includes a configurable information provider component called Grid Resource Information Service (GRIS) and a configurable aggregate directory component called Grid Index Information Service (GIIS). MDS-2 provides a simple instantiation of GIIS using a hierarchical structure. The simple directory accepts registration message from child GRIS or GIIS instances and merges these information sources into a unified information space.

The work that is closest to ours is by Adriana Iamnitch et al [7]. It proposes a mechanism for organizing the MDS-2 directories in a flat, dynamic P2P-network. Each Virtual Organization that participates in the grid dedicates a certain amount of its resources as *peers* which host information services and constitute a P2P- network between organizations. The authors evaluate several search algorithms based on query forwarding. Unfortunately, this approach exhibits some drawbacks similar to those of the Gnutella system [6]: the search is not deterministic and results not guaranteed.

JXTA search [12] uses a P2P network to support wide and deep search for rich information and targets at information search in general. The main problem it tries to address is the ability to search the huge amount of contents that cannot be handled by centralized approaches such as the one e.g., by Google.

Our work is based on recent DHT-techniques such as those in CAN [9]. The most important advantage of these P2P-systems as opposed to the earlier ones is that the new systems provide deterministic structure and performance guarantees in terms of logical hops; they also take advantage of network proximity. To our knowledge, it is the first approach that extends these methods for range queries. Also, our work is the first application of such advanced P2P-techniques for grid information services.

3. Distributed Indexing Infrastructure for Range Queries

In this section we first discuss how a collection of DHTs enhanced by the ability to query ranges can be used to provide search functionality in a grid information infrastructure. Then we describe the essential parts of the architecture of CAN [9] - the Content-Addressable Network - and explain how it can be extended to handle range queries.

3.1 Using DHTs for searching of resources

As mentioned above, parts of an information infrastructure such as GIIS organize the data about resources into a coherent image on which search can be performed. We discuss here how such a functionality can be provided by P2P-based techniques, offering the advantages listed above. Putting aside the issues of protocols and the architectures of current grid systems, we assume that each resource is described by a set of attributes with globally known types. A user interested in certain resources issues a query which is a combination of desired attribute values or their ranges; an example query in the CONDOR system

```
Requirements = Arch == "SGI" && OpSys ==
"IRIX6" && Memory >= 256
```

will search for all SGI machines running IRIX6 with at least 256 MByte of memory.

To further simplify we assume that in our setting each attribute has an independent indexing infrastructure (albeit

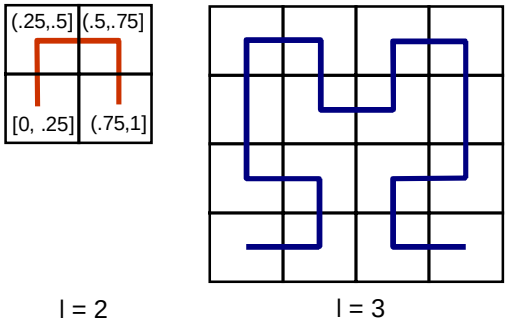


Figure 1: Hilbert curve for approximation level l = 2 and l = 3

other solutions as conglomerates of attributes or trees of attributes [14] are possible). In this scenario, for each attribute either a "traditional" DHT or the approach presented in this paper is used depending on its type. In the above example, the attribute "Arch" (and also "OpSys") could use "pure" CAN, Tapestry or Chord to manage a list of the pairs (architecture-type, machine-location). On the other hand, for the attribute "Memory" our enhanced DHT must be applied. To find resources specified by several attributes as in example above, the information infrastructure queries for each attribute present in the query the appropriate indexing infrastructure and concatenates the results in a database-like "join" operation.

3.2 Extending CAN for range queries

CAN organizes nodes into a P2P-network to solve the problem of data placement and retrieval over large-scale storage systems [9]. It uses a logical *d*-dimensional Cartesian space (a d-torus). This space is partitioned into zones, with a node (a peer) serving as owner of the zone. An object o is mapped to a point p(o) in the space (its "key"). A node R responsible for o is the one which has p(o) in its zone.To find R, we compute p(o) and contact any node in the network with a request to route a message to R. Routing from the contacted node to R boils down to routing from one zone to another in the Cartesian space.

Node addition corresponds to picking a random point in the Cartesian space, routing to the zone that contains the point, and splitting the zone with its current owner. Node removal amounts to having the owner of one of the neighboring zones take over the zone owned by the departing node.

In our context the "objects" are pairs (attribute-value, resource-ID), where attribute-value is a real number in R^1, and resource-ID is a handle to the resource. Without loss of generality, we assume that the attribute values are in range from 0.0 to 1.0.

A subset of the servers participating in grid will act as nodes in a (CAN-based) P2P-network and store the pairs (attribute-value, resource-ID). Each of them is responsible for a certain subinterval of [0.0, 1.0] of the attribute values. We call such a server an *interval keeper* (IK) and the corresponding subinterval its *interval*. Each server in the grid

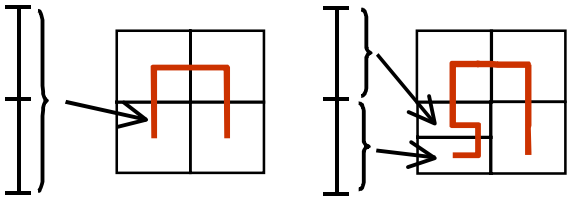


Figure 2: Adjusting the intervals dynamically

reports its current attribute value to an IK with the appropriate interval.

On the other hand, each IK owns a zone in the logical d-dimensional space. The mapping between the intervals and the zones is an essential element of efficient range queries. Intuitively, if two IKs have close-by intervals, then also their zones should be close-by. Furthermore, to allow an adaptation of the number of IKs to the total number of participating servers, even stronger property is required: if an interval I is split into intervals I_1 and I_2, then the zones of I_1 and I_2 must partition the zone of I.

Good candidates for such a mapping are the Space Filling Curves (SFCs). Especially, we use the Hilbert curve [1] for R^2 and its generalizations for R^d. The Hilbert curve is defined recursively: for an approximation level l = 1 it is a point, for l = 2 it looks like the left figure in Figure 1. For each higher approximation level we subdivide each zone into four subzones and "copy" into it a shrunk and possibly rotated version of the curve for l = 2 (see [1] for more details). Each such a created zone corresponds in a natural way to a certain subinterval of [0.0, 1.0]. For example, for l = 2 we have four equally sized intervals [0, 0.25), [0.25, 0.5) and [0.75, 1.0].

Given an attribute value, the mechanism that finds the IK whose interval contains this value is implemented as follows. We assume that the dimension *d* and the approximation level l are fixed and known globally. In the first step we compute the hypercube determined by the Hilbert function which corresponds to the interval containing our value. Then we contact any known IK and ask it to route our message to the IK whose zone encompasses this hypercube.

In practice, the distribution of nodes in R^d does not have to be uniform and static. Depending on the distribution of attribute values and the frequency of updates (determined by the rate in which the attribute changes), the distribution of the IKs can be dynamically adjusted. Figure 2 illustrates the refinement of a zone when the IK that owns the lower-left zone has to handle more updates because the population of resources that belong to that range becomes denser, or their attributes of those resources changes more frequent. In our current implementation we split an interval (and so also a zone) of an IK in two if it has to handle more than *L*, say, reporting servers. We call *L* the *split threshold*.

Allowing uneven distribution of IKs can affect logical routing performance since, for most of the P2P systems, the theoretical logical routing performance relies on the uniform distribution of nodes in the overlay. Fortunately, our

previous work provides techniques to take advantage of the network proximity and forwarding capacity of the nodes in overlay. In [13], we show that the routing performance can achieve on average 1.06 times of optimal routing performance.

4. Querying and Updating

In this section we first describe three implementations of the range queries, and in the second part the strategies for efficient updating of the data. In addition to simplicity, important design goals were the reduction of the network traffic for both querying and updating. In the case of range queries we also tried to minimize the query time, while for the updating strategies we wanted to ensure fault-tolerance and the self-organizing properties.

We say that an IK *intersects a query (range)* when its interval has a non-empty intersection with the query range.

4.1 Range query implementations

Common to all three implementations is the following method of routing to an IK which initiates the querying process: for a range query whose lower and upper bounds are l and u, we first route to the IK that owns the middle point $(l+u)/2$, and then have that node recursively propagate the request to its neighbors until all the IKs which intersect the query are visited ("flooding"). This method has the advantage of simplicity; it also works without changes if single attribute values are sought. Other methods are conceivable, such as initiating the flooding at the "ends" of the range as well; but since they are conceptually the same, we think that the proposed form exposes performance differences in the clearest way.

For the flooding part we propose three strategies. The first one is the most "naive" one, yet simple and applicable for different hash functions. The second and third strategy take advantage of the proximity preservation properties of the Hilbert function or its generalizations; the performance in these cases depends strongly on the choice of the hash function. We believe that those are the most natural approaches in our setting.

- **Brute force**: We first compute the smallest hypercube that encompasses all zones of the IKs intersecting the query, and then flood the request in a BFS-manner to all nodes which belong to this hypercube. Disadvantages of this method include wasted effort because not all nodes that are visited actually intersect the query.
- **Controlled flooding**: We let the current node forward the message to neighboring nodes that *definitely* intersect the query. An advantage of this approach is that no nodes that do not intersect the query are visited. One of the disadvantages is the fact that nodes may receive multiple messages for the same query. In addition, there is possibly less parallelism comparing with the brute force approach.
- **Directed controlled flooding**: The initial node starts two "waves" of propagation. In the first wave, the current IK propagates the query only to the neighbors that intersect the query *and* have "higher" interval than the current node. The second wave differs by the fact that the current IK propagates the query to neighbors with "lower" interval. An advantage is the reduction of message duplication as compared with controlled flooding.

4.2 Update strategies

The strategies described in the following ensure high fault-tolerance and self-organizing properties. Furthermore, we elaborate on them in order to minimize the network traffic. It is important to note that they can be applied independent of the used DHT-approach, and also do not assume the range query functionality described in the previous section.

In the "naive", unmodified case the servers report periodically the values of their attributes to the responsible IKs using the routing of the P2P-infrastructure. Each IK collects the pairs (attribute-value, server-id). If an update for one of these servers is not received in the next reporting round, the corresponding pair is erased.

In this way no additional activity is needed if the attribute value has changed so much that the pair (attribute-value, resource-id) is managed by another IK. This update strategy has two other advantages. First, the reporting servers do not need to know about splits or changes of IKs. Furthermore, if an IK fails, its replacement (created as in CAN) acquires complete and up-to-date information within one update round.

However, P2P-routing of updates incurs a lot of overhead compared to the physical network routing (IP-routing) in terms of network hops. In the following we describe approaches how to reduce this overhead.

Besides the techniques that we have proposed in our earlier work [13], that is, to introduce an auxiliary network to optimize routing, we can cache the IP-addressed of the relevant IKs. We assume that when an IK received an update, it sends back a message containing its IP-address and the limits of its interval. The server records this information in a cache of a fixed *cache size* s, say. In the subsequent update rounds it checks first the cache for an IK whose interval contains the current attribute value. If found, the update information is send to this IK via IP-routing. In this scenario, the P2P-routing is used only at a cache miss or if the IP-routing failed, i.e. if there was no acknowledgement from the IK.

This strategy might become less efficient if the number of IKs grows and so the sizes of their intervals decrease. The following approach turned out to be very efficient in avoiding this effect and further increasing the share of IP-routing. When searching in the above-mentioned cache, we artificially "broaden" the intervals of the cached IKs by subtracting t, say, from the lower interval limit and adding t to the upper limit. The parameter t is called the *tolerance level*. This method can be only applied if it is not necessary to know the exact attribute values.

Finally, under the same assumptions we can use the following method to balance the load between IKs. Each time

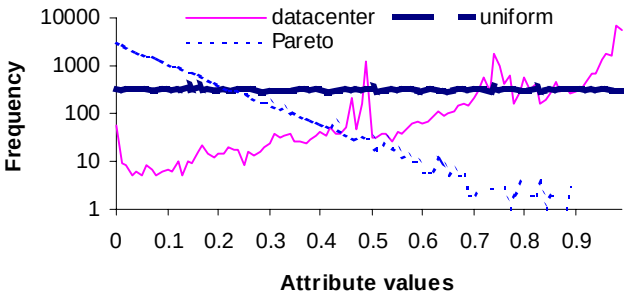


Figure 3: Densities of the used attribute values

Parameters	Range
# of IKs (N)	100, 500, 1000, 5000, 10000
Dimension d	2, 4, 8
Attribute Value Distribution	Synthetic: uniform and Pareto
	Real-life: data center traces

Table 1: Common parameters

when a P2P-routing is used, the reported attribute value is randomized in less significant digits by adding a random value from an interval [-*r*,*r*]. The parameter *r* is called the *randomization level*. This causes the update information to be stored on an IK with an interval "close" to the true value. It helps if a lot of very similar values are reported.

5. Evaluation

To show the effectiveness of the techniques, we evaluate them via simulation. We assume that the attribute used in the experiments is current processor capacity. Thus, in queries we search for all machines with processor capacity in a certain range.

Table 1 summarizes common parameters for our experiments. One of then is the distribution of attribute values. We use three different distributions shown in Figure 3: uniform random, normalized Pareto with parameter a = 100, and traces of residual server processing capacity from a data center. The traces come from 41 servers and were collected in 5-minute intervals during 34 days starting on September 2nd, 2001 (see [10] for more details). It is interesting to see that a Pareto and the data center distributions are somehow similar (albeit reversed and the data center distribution have several peaks). This confirms a frequently made assumption that a Pareto is a better fit for real-life utilization patterns than the uniform distribution.

5.1 Query performance

The preprocessing step for the following experiments is the building up of simulated CAN networks. For each parameter set we start with one IK and report a set of attribute values (with a specific distribution) until the number of IKs created due to splits reaches a specified value. The parameter *L* is 20. In case of the data center traces we concatenate traces from different days and times of the day to reach a sufficient number.

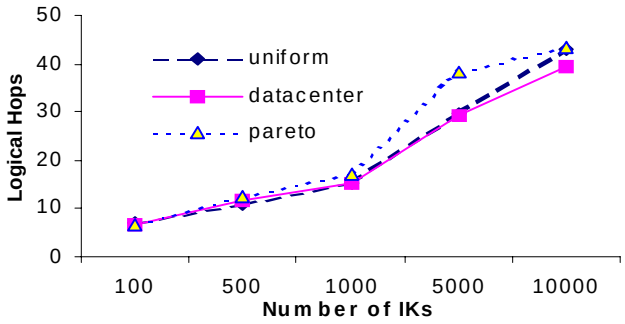


Figure 4: Comparison of logical routing performance, d = 2

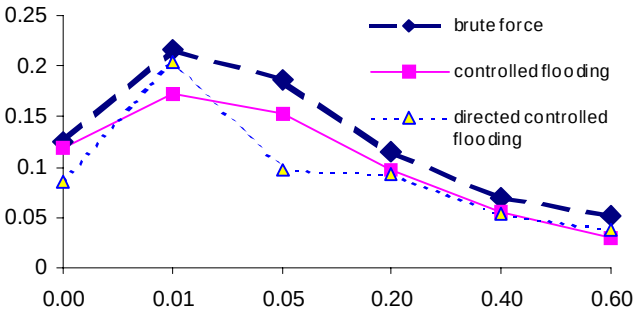
In the first set of experiments, we study the query performance with a simulated CAN network. The time taken to perform a range query consists of (1) the time to route to the IK that owns the middle of the range, and (2) the time spent to flood the query request to all the nodes intersecting the query.

To measure (1) we randomly select a pair of nodes in the network and route between them. Figure 4 shows the number of logical hops needed between a random pair of nodes averaged over 10 times the number of nodes of experiments (e.g., if the number of nodes is 100, the number of experiments is 1000). The x-axis shows the number of IKs and the y-axis shows the average number of logical hops.

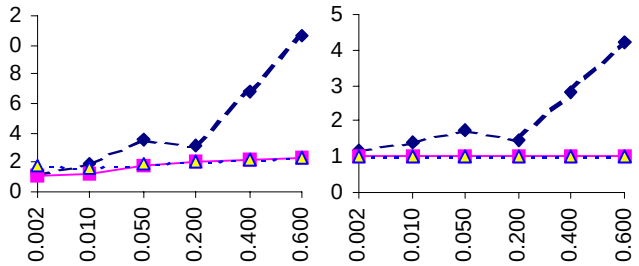
From this figure we see that node distribution can slightly affect routing performance for CAN when d = 2. But the differences are much smaller for CAN with a higher dimension such as 4 and 8. (Due to space constraints we omit the cases d = 4 and 8.) For d = 8, the average number of logical hops between two randomly selected nodes ranges from 5 to 8 for the number of IKs used in the system. In fact, with the techniques described in [13], we can achieve close to optimal routing performance.

To measure (2) we compute the longest and the average path length taken to reach all IKs that intersect the query, the number of messages sent, and the number of nodes that are visited during the flooding. These quantities are normalized according to the number of IKs which intersect the query (note that this also applies to the average path length, so that in the figures the numbers on the y-axis are less than 1). We show results averaged over 10 random tries for ranges which vary from 0.002 to 0.6. The longest and the average path length provides indication how fast the query request can be propagated to the nodes intersected by the query.

Figure 5 to Figure 7 illustrate the results for the three different distributions with 1000 IKs and d = 2. We only show the average path length since the shapes of the curves for longest path and average path are very similar. In these figures, (a) compares the average path length of the three dif-

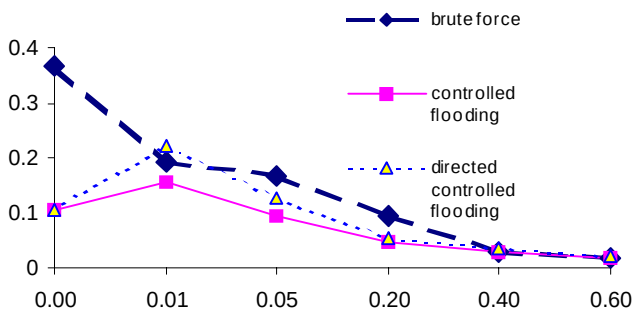


(a) Average path length

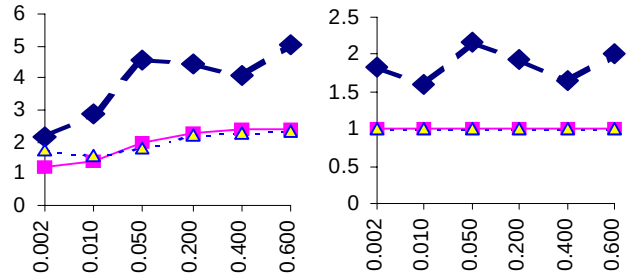


(b) Number of messages (c) Nodes visited

Figure 5: Average path length, number of messages, and number of nodes visited, where x-axis is the size of range query. The figures are for the data center trace, N = 1000 and d = 2.



(a) Average path length



(b) Number of messages (c) Nodes visited

Figure 6: Average path length, number of messages, and number of nodes visited, where x-axis is the size of range queries. The figures are for the uniform distribution trace, N = 1000 and d = 2.

ferent query propagation strategies, (b) compares the number of messages, whereas (c) compares the number of nodes visited. We can observe the following:

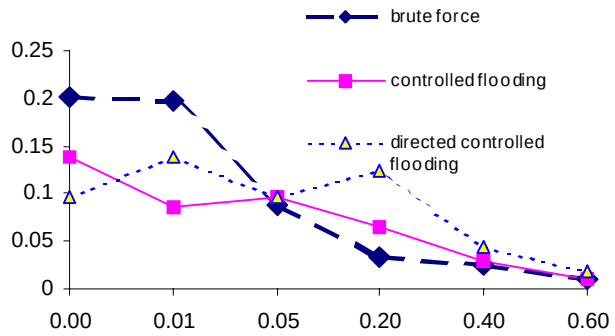
- The controlled flooding algorithms (undirected and directed) perform better than the brute force approach with respect to the time taken to propagate the query requests (longest and average path length). The only exception is for the Pareto distribution, where the controlled flooding algorithms perform slightly worse for query ranges 0.2 till 0.6. This is probably due to the non-uniform distribution of the IKs in the system.
- The controlled flooding algorithm performs consistently better than the brute force algorithm with respect to the number of messages and number of nodes visited. The improvements are more significant for queries with relatively large ranges. In fact, for the two controlled flooding algorithms, only those IKs whose intersect the query are visited.
- The directed controlled flooding algorithm reduces the messaging overhead only slightly. For d = 2, the performance of the two controlled algorithms are comparable.

Figure 8 shows the same set of curves for d = 4, the data center trace, and N = 1000. The situation for d = 8 is very similar. We observe that for a higher dimension, the controlled flooding algorithms perform consistently better than the brute force algorithm in terms of number of messages and number of nodes visited. The performance improvements are more significant for than those for a low dimension. Concerning the time to propagate the query request, the performance differences for queries with smaller range are larger than queries for large ranges.

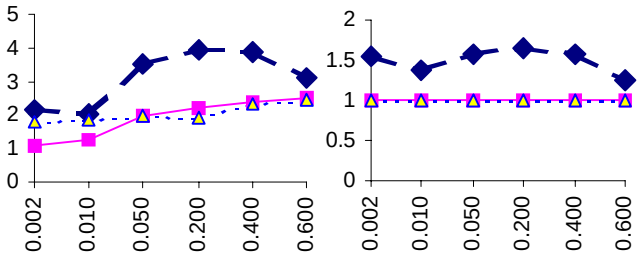
5.2 Update efficiency

In the following we evaluate the effects of the strategies described in Section 4.2. For all experiments the data center traces were used. We simulated updates occurring in 5-minute intervals during a single day (this gives 288 update rounds). To increase the number of updates per round, we concatenated the data for a single 5-minute "slot" of a 24 hours period from 33 days. We set 50 as the threshold at which the interval of an IK is split. At each update round new splits could occur (which caused the number of IKs to increase). To decrease the number of parameters, we did not merge together neighboring IKs with low number of reporting servers. The number of IK after the last round was in the range 110-160.

Figure 9 provides intuition how the changes in update patterns may affect the routing efficiency. The efficiency is measured as a ratio of IP-routed update messages to the total number of update messages. In round 97, obviously a new "pattern" of the update values occurs. It causes many cache

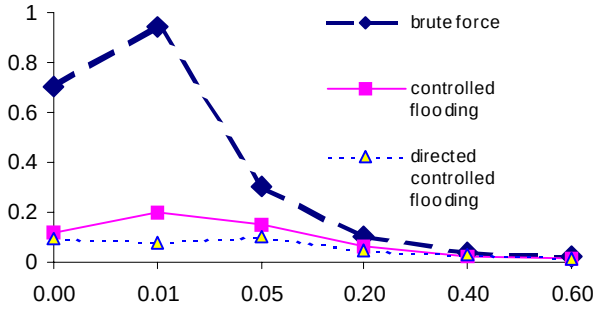


(a) Average path length

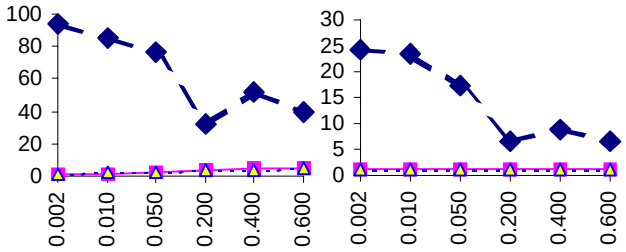


(b) Number of messages (c) Nodes visited

Figure 7: Average path length, number of messages, and number of nodes visited, where x-axis is the size of range queries. The figures are for the Pareto distribution trace, N = 1000 and d = 2.



(a) Average path length



(b) Number of messages (c) Nodes visited

Figure 8: Average path length, number of messages, number of nodes visited, where x-axis is the size of range query. The figures are for the data center trace, N = 1000 and d = 4.

failures at the reporting servers, and thus a drop in the fraction of IP-routed updates. An additional effect is the likely creation of new IKs, so further cache entries become outdated. The same figure shows that the tolerance level t indeed drastically improves the routing efficiency. This is discussed in the following

Figure 10 visualizes the effect of the servers cache size s and the tolerance level t on the fraction of all IP-routed updates in relation to all updates (summed over all rounds). We see that a value of 10 for the cache size s (which corresponds to 7-10% of the number of IKs) is close to the cache saturation level. More significant is the influence of the tolerance level t. For s = 10 already a small value of t = 0.001 increases the fraction of IP-routed updates to a level not attainable by the means of increasing the cache size in case t = 0.0. For t = 0.01 already a small cache size ensures a high ratio of IP-routed messages.

In Figure 11 we see the effect of the tolerance levels on the number of created IKs per round. With higher tolerance levels, less and less of new IKs are needed during the successive update rounds.

Figure 12 shows the effect of the randomization level r on the density of the number of IKs per interval. With increasing value of r we expect this density to be more evenly distributed. It turns out that this effect is not very pronounced, at least not for this data set. Therefore this approach seems to have only a limited value for balancing of load between the IKs. However, for a peak attribute density at 0.7 a positive effect can be observed (also cf. Figure 3). This leads to a conjecture that for degenerate update value sets (e.g. many identical values) this approach performs well.

6. Conclusions and Future Work

In this paper we investigate the issue of extending P2P-based Distributed Hash Table systems such as CAN to allow range queries. To our knowledge, this is the first work that addresses this problem.

We propose three simple strategies for propagating range-query requests, and strategies to minimize the communication overhead during the attribute updates.We evaluate the effectiveness of these strategies through simulations using both synthetic and real-life workloads. We show that the techniques we propose are effective in meeting the goals of scalability, availability and communication-efficiency. The design of the system and its evaluation targets its usage as a part of the information infrastructure for grids.

While this system is based on CAN, our future work will include evaluating of similar approaches which are based on Plaxton-like DHT-systems such as Tapestry or Chord. We also intend to investigate application of caching and replication to further improve reliability and availability of the information service. An interesting problem is handling of

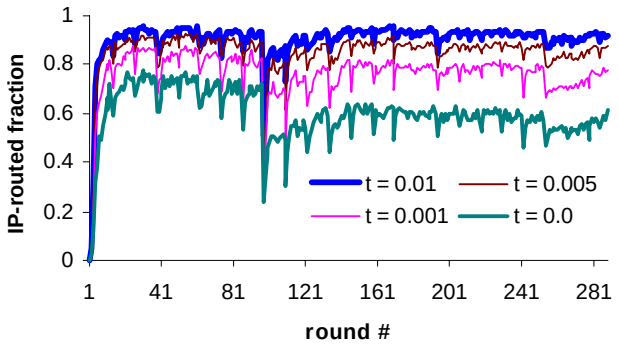


Figure 9: Example how the changes in update patterns affect the routing efficiency

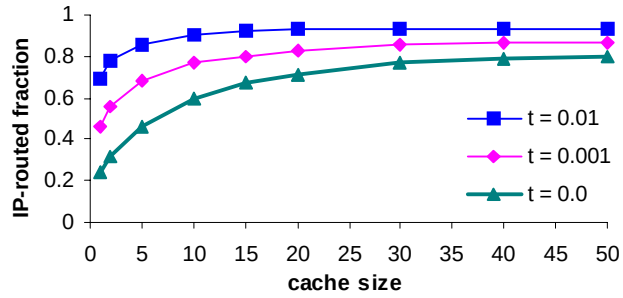


Figure 10: Influence of cache size and tolerance level t on the fraction on IP-routed updates

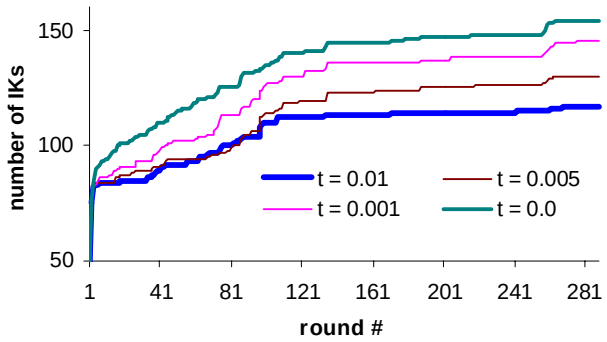


Figure 11: Effect of the tolerance level t on the number of IKs

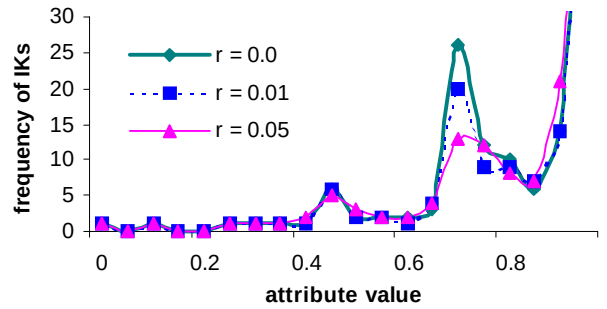


Figure 12: Effect of the randomization level r on the distribution of IKs density

multiple attributes by a *single* enhanced DHT-system, as well as coupling these techniques with a real grid information infrastructure.

7. References

[1] T. Asano, D. Ranjan, T. Roos, E. Welzl, and P. Widmaier, "Space Filling Curves and Their Use in Geometric Data Structures", *Theoretical Computer Science*, 181, 1997, pp. 3-15.

[2] I. Clarke, O. Sandberg, B. Wiley, and T.W. Hong, "Freenet: a Distributed Anonymous Information Storage and Retrieval System", *Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, USA, 2000.

[3] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10)*, IEEE Press, 2001.

[4] F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with CFS", *SOSP'01*, Banff, Canada, 2001.

[5] C. Decusatis, "Grid computing: the next (really, really) big thing - interconnecting millions of distributed-computing devices", *BYTE*, Spring 2002.

[6] Gnutella, http://www.gnutella.org.

[7] A. Iamnitchi, and I. Foster, "On Fully Decentralized Resource Discovery in Grid Environments", *International Workshop on Grid Computing*, Denver, Colorado, 2001.

[8] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An Architecture for Global-Scale Persistent Storage", *ASPLOS '00*, MA, USA, 2000.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network", *ACM SIGCOMM '01*, San Diego, CA, USA, 2001.

[10] J. Rolia, X. Zhu, M. Arlitt, and A. Andrzejak, "Statistical Service Assurances for Applications in Utility Grid Environments", submitted, 2002.

[11] 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 '01*, San Diego, CA, USA, 2001.

[12] S. Waterhouse, "JXTA Search: Distributed Search for Distributed Networks", Sun Microsystems, May 2001.

[13] Z. Xu, M. Mahalingam, and M. Karlsson, "Turning Heterogeneity to an Advantage in Overlay Routing", *HPL-2002-126*.

[14] Z. Xu, and Z. Zhang, "Building Expressways for P2P", *HPL-2002-41*.

[15] Z. Zhang, M. Mahalingam, Z. Xu, and W. Tang, "Scalable, structured Data Placement over P2P Storage", *HPL-2002-40*.