

Multi-set DHT for interval queries on dynamic data

Georges Da Costa

Georgio.da.costa@gmail.com

Salvatore Orlando,

orlando@orlando@dsi.unive.it

Information Science and Technologies Institute

Italian National Research Council

56100 Pisa, Italy

Marios D. Dikaiakos

mdd@cs.ucy.ac.cy

Computer Science Department

University of Cyprus

POBox 20537

CY1678 Nicosia, Cyprus



CoreGRID Technical Report
Number TR-0084

March 27, 2007

Institute on Knowledge and Data Management

CoreGRID - Network of Excellence

URL: <http://www.coregrid.net>

Multi-set DHT for interval queries on dynamic data

Georges Da Costa

Georgio.da.costa@gmail.com

Salvatore Orlando,

orlando@orlando@dsi.unive.it

Information Science and Technologies Institute

Italian National Research Council

56100 Pisa, Italy

Marios D. Dikaiakos

mdd@cs.ucy.ac.cy

Computer Science Department

University of Cyprus

POBox 20537

CY1678 Nicosia, Cyprus

CoreGRID TR-0084

March 27, 2007

Abstract

Scalability is a fundamental problem for information systems when the amount of managed data increases. Peer to Peer systems are usually used to solve scalability problems as centralized approaches do not scale without large dedicated infrastructure. But most current Peer to Peer systems do not take into account that indexed data can be dynamic and change their values very often. Thus, we propose the *Multi-set* approach, which aims to find the best trade-off between DHT-based network and total replication. This approach is built over classical DHT Peer to Peer system. It can improve most of pure DHT Peer to Peer system by taking into account the dynamism of resources. Evaluation is done by modeling, simulation and experimentation on PlanetLab. This approach is more efficient than DHT Peer to Peer system and total replication whichever the dynamism of resources is.

1 Introduction

Grids are based on several basic but nevertheless necessary services. One of such services is the resource manager. This service has to keep track of the Grid state and has to be able to locate resources corresponding to user queries. The attributes associated with the diverse physical resources making up the Grid can be of various types. They can be static, such as the type of network card, or dynamic, such as network bandwidth. Some attributes can be characterized by an intermediate dynamism, such as the number of free processors in a cluster.

Resource managers have thus to track the various Grid elements and characteristics. To this end, they have to manage large amounts of information. Information is constantly produced in every node of the Grid. Resource managers have to answer to queries which are initiated by users willing to run their applications on the Grid. Users submit their queries to portals. Those portals are widely distributed as there are at least one per community. The burden

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

and complexity of resource management should not increase too much with the scale of the Grid system. Moreover, such a system should be able to efficiently answer queries [14] such as: *find clusters with at least 32 free processors and Ethernet network*.

Difficulties arise due to the large scale of current Grids. The centralized approach shows its limits, since it sacrifices data freshness for efficiency. For example, a system using a tree structure (like the predominant *Monitoring and Discovery System* of Globus) has to increase timeouts to prevent overload of the root. The physical resources, a set of clusters in this case, are the leaves of the tree. Information on the leaves is updated every second. Leaves are aggregated in groups (sub trees), and so on until reaching the root which aggregates all resource information of the system. For large systems, the root is updated every ten minutes or more to reduce its workload [4].

Peer to Peer systems are known [10] for having solved the file-sharing scalability problem and for being fault-tolerant and easy to deploy and manage. However, classical Peer to Peer systems (Chord, Freenet) cannot directly be used to solve the problem of Grid resource management, as they lack key functionalities, like the ability to give several answers for one query or to answer complex queries.

Peer to Peer systems for Grid resource management have been already introduced [2, 5]. Even links between databases and Peer to Peer have been explored [7]. However, most of these systems are not yet efficient enough for our purposes: they are not reactive, and in some cases, use a number of communications proportional to the number of clusters in the system. Secondly, most of these studies do not take into account the variability of stored information.

Our goal is to improve current systems [2, 5] using a Peer to Peer approach to provide dynamism aware resource management. Users ask about particular resources such as: *Where can I find computers with exactly 100Mb/s network*, or range of resources such as: *Where can I find storage with at least 1To free* or mix of these. Queries are often kept simple because the most versatile the query is, the less optimizations are possible. For this reason, resource attributes in Grids are usually represented as (or can be transformed in) integer numbers. Thus queries submitted by users can be easily translated[5] into disjunction of conjunction of simple queries about particular resources or range ones. The Grid resource manager can process those simple queries independently and then merge the results. This model of queries is quite generic as it is possible to translate basic LDAP or XPath queries (used by Globus) to this model. Thus a Grid resource manager can be built using a simple module that answer interval queries, possibly implemented as a distributed service. This basic module shall share the qualities of the whole system: managing dynamic data, answering fully distributed queries and updates, and being scalable. Our proposal of such a system is the *Multi-set DHT* which is optimized for all types of information dynamism.

In the following, we will first evaluate the performance of other systems, then we will present our Multi-set approach, and finally we will evaluate its performance.

2 Interval queries

In this study, we focus on systems that can manage interval requests on dynamic objects efficiently.

2.1 Requests

Interval queries are used to answer inquiries about resources [5], such as: "Find resources of type *cluster* with at least *32 free processors*, *SCI* or *Myrinet Network*, and *nfs_version* of at least 4". Such an inquiry can be translated to three simple queries about resource-attribute values:

- two interval queries
 - *Free Processors* > 32
 - *nfs_version* > 4
- one exact query
 - *Network* = *SCI* or *Myrinet*

There are several ways to execute a generic resource inquiry using simpler interval queries [5]: One approach would be to process the simple queries in parallel and then merge their results. Alternatively we could process only one of the simple queries and then check which elements of the answer fulfill the initial inquiry. Boolean expressions

of simple queries typically correspond to most of the real usage of Grids. Nevertheless, they prevent users from doing second-order queries such as : "Find a cluster where there are as many computer elements as storage elements".

Moreover we analyzed logs of real Grids (EGEE [1], CiGri [3]), and such complex queries were not used. The data of EGEE concern the period from 14/02/2006 to 30/01/2007 and are of the South-East nodes of EGEE. Logs shows 144389 queries, of which 1500 are queries number of processors, and 1334 are about amount of memory. For Icluster2, which is a French cluster (Grenoble) using the OAR cluster manager, logs shows 279834 queries from 04/05/2004 to 18/11/2005, of which 5% are only for processors number and 300 are linked to the network type.

In the following we will concentrate on the problem of resolving interval queries and updates on a single resource attribute. The way of managing more complex queries using such basic component can be found in [5].

To this end, we provide a distributed index for each attribute which manages a distributed store of attribute values (*keys*) and pointers to grid nodes (*object*) that provide access to corresponding resources. Assumptions on the objects are : Each object is associated with exactly one key. Keys may change over time. Each query can be either for one key, or for an interval of keys. Each key can be associated with any number of objects. For example, for the *Free Processors* attribute, the key 32 is associated with all the clusters that have exactly 32 free processors. The attribute value is considered as being an integer between 0 and *Max*. Thus the key space is $[0, Max]$.

In such a system, the main functionalities are to update the key associated with a value, and to find the object associated with a particular value or a range of values. The minimal interface of a system usable as a basic component for resource management should be :

- object list = **get**_{attribute}(*bound*₁, *bound*₂);
- object list = **get**_{attribute}(*key*);
- **update**_{attribute}(*oldkey*, *newkey*, *object*);

The first query returns all the objects in the system whose keys are between *bound*₁ and *bound*₂. Several objects can be found at the same key. Updates are done independently: For example to change the available download bandwidth of a cluster : `updatedownload_bandwidth(1000, 650, icluster.imag.fr)`.

Building such a system without dedicated infrastructure is complex. It needs to react fast to changes to reduce the false information given to users. Moreover such a system must manage peers (nodes that participate in the system) that come and go, due to crashes or simply to local changes of policy.

2.2 Distributed Hash Table (DHT)

DHT Peer to Peer networks are distributed indexes that link integer keys with single objects. Most DHTs efficiently answer simple queries like: which peer is responsible for a particular key. Typically, the worst case cost to locate an object, in terms of number of messages exchanged, is logarithmic with the number of peers. For example, Chord [13] guarantees that queries need less than $\log(\text{participants})$ messages. The mechanism of Chord is simple: each peer has a randomly chosen identifier (*id*) and each indexable object has a *key* obtained by hashing its content. Identifiers and keys belong to the same integer value space. To avoid collisions the space is usually chosen to be large. Current implementations use $[0..2^k - 1]$, with $k = 80$, i.e. 80 bits. Each object is officially managed by one peer whose *id* is the nearest but smaller or equal to the object's *key*. It is this peer that needs to be contacted to obtain the object associated with the *key*. Thus, each peer stores the objects whose keys are between its *id* and the nearest superior *id* of a peer (the peer with this *id* is called its successor). Each peer only knows (Figure 1) a handful of other peers (neighbors) whose *id* are the closest to : $id + 1, id + 2, id + 4, \dots, id + 2^i, \dots, id + 2^{k-1}$ (modulo 2^k). To locate an object, a request is routed according to its value (content-driven routing). On each peer, the request is forwarded to the neighbor which *id* is the nearest to the *key* (Figure 2). The request is forwarded by the same method until it reaches the peer storing the requested object. At each forward, the number of peers between the current peer and the destination distance is halved, so there are at most $\log(n)$ messages to contact the right peer. This performance is obtained with only a partial knowledge of the global system on each peer.

Classical Peer to Peer systems are too limited to be directly used for resource management. First, they can only link one object to each *key*. Secondly they can only answer queries about one single *key*. Some systems are using a Peer to Peer approach to address the interval query problem: for instance Probe [12], Baton [8], and others [9, 6] described in [14]. These systems use the range technique (described bellow) to provide interval functionality. In this case, each peer still manages all the objects with keys between its *id* and the *id* of the next peer. However, each key can

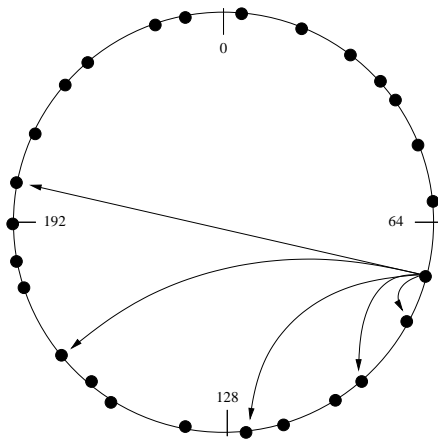


Figure 1: Each peer in Chord has a random id between 0 and $2^k - 1$. Each of them knows only a small number of neighbors. Each peer is responsible for the object whose id is between its own id and the id of its successor.

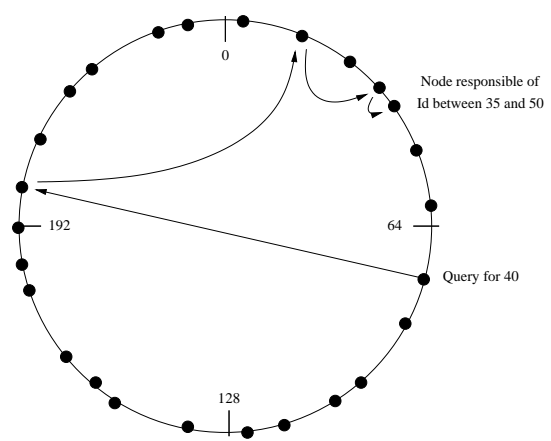


Figure 2: To find an object in a Chord system, one peer contacts the peer which is nearest to the object id. The query is forwarded until finding the peer responsible for the object.

	query	update	join/leave
Probe[12]	$N^{1/d}$	$N^{1/d}$	large
Nikos05[9]	$\log(n)$	$\log(n)$	$\log(n)$
Baton[8]	$\log(n)$	$\log(n)$	$\log(n)$
Ganesan04[6]	$\log(n)$	$\log(n)$	$\log(n)$

Figure 3: Comparison of several Peer to Peer systems able to manage intervals. Results for retrieving objects are given for small intervals.

be associated with several objects: In classical Peer to Peer systems a key is only a sequence of bits with no meaning in itself. Here keys can have a defined meaning such as being a number of processors and queries can be done on this value. The key of an object can be the value of one of its attributes for example. The key can be the free disk space for objects like storage servers for instance. Those systems have the same performance as usual DHT for a single key (Figure 3).

When a user wants to obtain the objects whose keys are between $bound_1$ and $bound_2$, the algorithm of the range query is as follows (see Figure 4):

- Contact the peer responsible for $bound_1$ using a Peer to Peer algorithm.
- This peer forwards the query to its successor.
- This forwarding is done until reaching $bound_2$.
- The peer responsible for $bound_2$ sends the aggregated answer to the user.

The first problem concern the query cost. For small intervals, which are spread on a few peers, the cost remains the same as classical DHT, i.e. logarithmic. However, the query cost of those systems is clearly linear in the size of the interval. Indeed the keys are spread amongst all the peers, so the larger the interval query is, the more peers it involves.

The second problem is related to the workload distribution as, for some attributes one of the two bounds is always open. For example, queries for the attribute *number of free processors*, users typically ask for clusters with *at least* a certain number of free processors. For *open* interval queries, the peer responsible for one of the two extremities of the key space has to answer all the queries. For example, since typical queries looking for clusters with free processors are interval queries that are open on the right, the peer responsible for the right extremity of the key space has to answer all the queries.

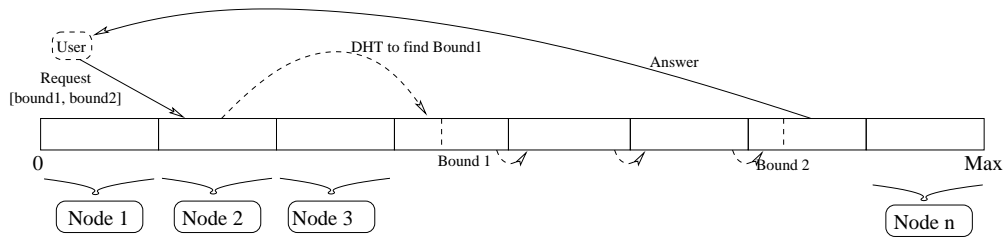


Figure 4: For Peer to Peer systems able to answer interval queries, the global space is split in ranges, one for each peers. To find an interval, one has to search for the peer responsible for the interval's lower bound, using the global DHT. Then this peer contacts its neighbor, and so on until the upper bound of the interval is reached.

Updates are easier to manage. In a system where the key has some meaning (like being the value of an object attribute), there is no update by interval. The update consists in changing the value of the *key* in the tuple (*key*, *object*). The simplest way is to do first a request of removal of the old tuple, followed by an insert of the new one. Each of these two operations are based on the query of the location of the old and new peers owning respectively the old and new *key*. So, the update cost is logarithmic.

To summarize, the cost for open queries is linear with respect to the number of peers, and the cost of single updates is logarithmic. In conclusion, using one DHT appears to be efficient only when most of requests are updates.

2.3 Total replication

Another extreme method would be to adopt a total replication of all the objects to be indexed amongst the peers. The query is free in terms of messages exchanged as the query can be processed locally. But the changes of resource attributes becomes very expensive since all the peers must be contacted to update the indexed information. This method is perfect when there are a lot of queries and a very small number of updates as cost for updates is linear, and query cost is constant.

2.4 Meta-methods

A number of DHT systems have been proposed in recent literature [8, 9, 12, 6] to address the resource management problem. Each of these systems provide a particular algorithm for handling interval queries.

To improve those systems, one can use them as a black box. The characteristic of this black box is that it can answer interval queries. By building a system on top of it, it is possible to improve the use of this black box. This high level method can be used to improve any system providing the same interface independently of the implementation.

An example is provided by Sword [5] which shows that using one of these black boxes for each attribute of a resource improves the overall performance compared to using one module for all the attributes.

3 Mutli-set approach

In this section we discuss our proposal to manage objects of dynamic characteristics. In the case of grid resource management, the number of free processors in each site is dynamic. In this example, the *object* is the site URI¹, and the *characteristic* is the number of free processors of the site.

Our Multi-set DHT can be thought as a system that aims to find the best trade-off between the two opposite approaches discussed earlier: the single DHT-based Peer to Peer network that spans across all the peers of a Grid system, and the case where the resource index is fully replicated to all Grid peers.

Grid resource management is based on information made available by each resource. In the following we consider that *peers* of our Peer to Peer resource discovery system are the same computers that manage the local resources of the Grid sites. Usually, each cluster of the Grid uses a local resource manager. Such peer is usually a computer linked to the computing and storage elements. This allows us to obtain a fully decentralized and self-managed system. Each

¹Universal Resource Identifier, used to contact a site

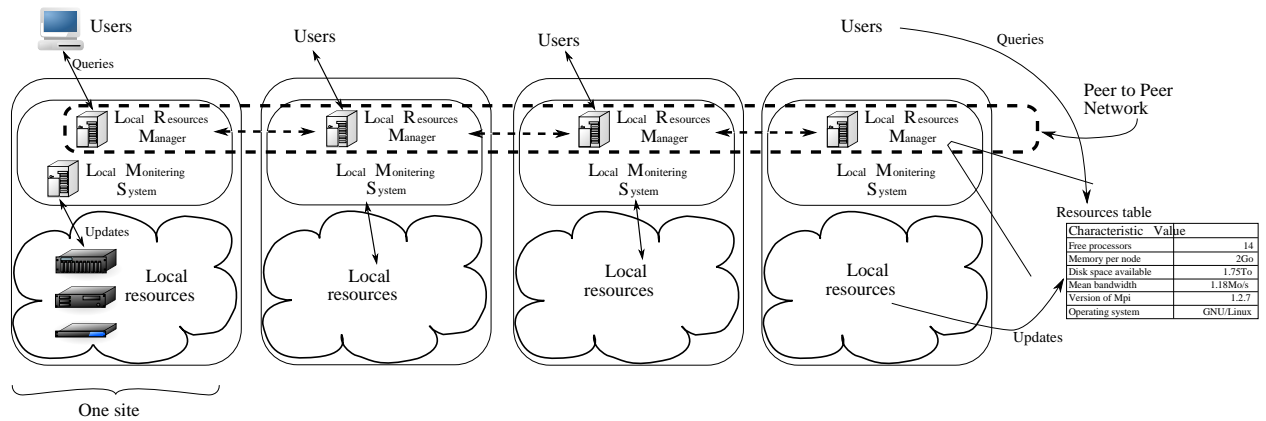


Figure 5: Each site uses at least one Local Resource Manager. This *lrm* receives the updates concerning the resources it manages locally. It receives queries from users too. When put together in a Peer to Peer network, they forward queries and upgrades according to local algorithms.

user using our system is connected to one of these peers. As a simplification we will only consider the user and the peer he uses as one single entity. A diagram of this structure is shown in Figure 5.

We will use the term *request* for either update or query. We assume that each resource attribute has its own *ratio of update/request*. As seen in the previous section, classical DHT and total replication are efficient for only a few ratios (when queries are open ones) : Total replication is efficient for ratios close to 0 and one DHT spanning all the peers of the system for ratios close to 1.

The goal of our Multi-set DHT approach is to provide a transparent system that is able to achieve the best performance possible whatever the ratio is, event if the ratio evolves. To this end, we synchronize several classical DHT, each supported by a distinct *set* of peers. Each peer is in one single *set*, and all *sets* are approximately of the same number of peers.

3.1 Sets

Sets are disjoint groups of peers (Figure 6). Sets are a partition of all the peers. Each of these sets behaves like an interval-enabled object management system. All these sets manage the same information. Thus all (*key*, objects) are duplicated on each set. With this property, a query can be done on any set and will return the same value, independently of the chosen set.

In order to keep the sets synchronized, updates are to be done on each of them. To this end, at any time, each peer keeps at least a reference to a random peer of each set. When an update is issued on a peer, it is processed locally on the set and, at the same time, it is sent to the other sets to keep them up-to-date.

To distribute evenly the workload on all the peers, these references are used for queries too. When a query is issued on a peer, the peer chooses randomly a set to process it.

In the following the sets are implemented as interval enabled DHT systems.

The two fundamental operations provided (Figure 6) are :

Update : In this system, an update is done by contacting an element of each set and using the usual update system of the set.

Query : A query is done by first querying a peer of a randomly chosen set and then using the usual query system of the set.

Intuitively, the update cost increases with the number of sets as there are more sets to inform of the change. In contrast, the query cost decreases as the query is solved inside one set, and the size of each set is reduced when the number of sets grows.

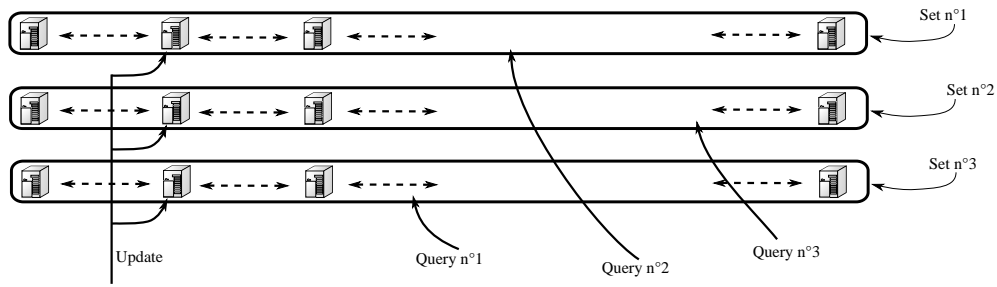


Figure 6: Multi-set system: Objects are replicated across several groups of peers called *sets*. An *Update* is forwarded to all the sets. A *Query* goes to a random set. The number of sets depends on the ratio Query/Update

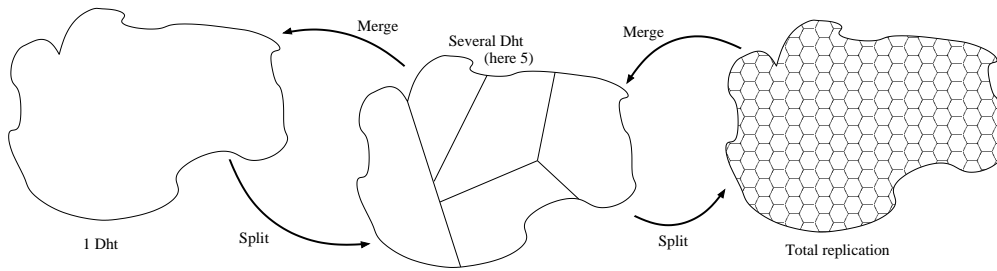


Figure 7: On the left side, all the peers belong to the same DHT. On the right one, all the peers have a copy of the same index, i.e., the replication is total. One of the possible intermediate cases is shown in the center, where peers are subdivided into several sets. Each peer set supports a distinct DHT. In order to switch between a system configuration and another one, *merge* and *split* functions are used.

3.2 Trade-off

Figure 7 shows the Multi-set DHT approach. All the peers are distributed in *sets*. Each of these sets is a DHT as described above. If there is only one set, then this corresponds to a *single DHT* case, whereas if there is only one peer per set, this corresponds to the total replication case.

The Multi-set approach is to be between those two extremes. As the ratio Update/Requests evolves over time, it is necessary to have a system where *update* and *query* costs can be adapted depending on its use.

With this system, *Updates* become more costly with increasing the number of sets, but *Queries* become less. By adapting the number of sets to the ratio Query/Update, a good trade-off can be achieved. It should distribute the workload more efficiently too.

Coordinators obtain information from peers to extract the current ratio. Then they adapt the number of sets by using two methods : *Merging*, which decreases the number of sets, and *Splitting*, which creates a new set.

3.3 Model

A first step to evaluate the quality of this approach is to model it and compare to the other two extremes approaches.

To compare them, the metric used will be the mean number of messages used for answering a request. This metric is relevant as it gives information at the same time on the latency and on the resources used by the system.

As the underlying Peer to Peer systems can adapt to spread requests amongst the peers [15], we assume that requests are uniform in the model. Requests will be issued uniformly amongst the peers. Updates will change a random value to another random one. Queries will be open on right, and the left bound will be uniformly distributed.

Let N be the number of peers in the system, p the number of sets, α the ratio of updates. The probability that a generic request is an update is α , while the probability that it is a query is $1 - \alpha$.

Peers are supposed to be uniformly distributed amongst the key space.

We use the chord underlying systems which uses $\ln n$ messages on average in a n peers system to locate an object using its key [13].

Update cost for one DHT Two operations are done, first remove the old value, then insert the new one. The cost is thus:

$$2 \ln N$$

Query cost for one DHT As peers are uniformly distributed, the cost for one query is proportional to the size of the interval. The mean size of the interval is half the key space. To answer a query regarding half the key space, half the peers needs to be contacted. The cost of contacting the first bound is $\ln N$. The cost is thus:

$$\frac{N}{2} + \ln N$$

Query cost for total replication As the object is already local there is no communication in this case.

Query cost for total replication All the other peers are contacted leading to a cost of:

$$N - 1$$

Update cost for p sets The update is done on the current set (of $\frac{N}{p}$ peers), then on the others ones. Beside the real updates, $p - 1$ communications to contact the other sets are needed. Finally the cost is:

$$2p \ln\left(\frac{N}{p}\right) + (p - 1)$$

Query cost for p sets There is only one set that solves the request. There is a probability of $\frac{1}{p}$ to randomly choose the local set and thus to prevent one communication between the sets. The cost is:

$$\frac{p - 1}{p} + \frac{N}{2p}$$

Request cost for p sets Consequently the mean cost of a request in the Multi-set system is:

$$\alpha * (2p \ln\left(\frac{N}{p}\right) + (p - 1)) + (1 - \alpha)\left(\frac{p - 1}{p} + \frac{N}{2p}\right)$$

This function has only one minimum with $p > 0$.

The curves on figure 8 show the three costs. For the Multi-set, for each α , the p used is the one that minimizes the mean cost.

3.4 Algorithm

For each ratio it exists an optimal number of sets that minimize the number of messages in the system. This number depends on the number of peers too.

As those values evolves over time, it is not possible to define them one and for all before launching the Multi-set. Thus is must be able to evolve towards the optimal number of sets when running. To achieve this it needs first to evaluate the right number of sets, and second, to change the number of sets.

The three basic operations on which this system is based are the following.

Decision which chooses when to do a *Merge* or a *Split*.

Merge which reduces the number of sets by one.

Split which increases the number of sets by one.

Except during a split, all the peers know at least one peer of each other set. Decisions are not to append frequently beyond initialization as pattern of use do not often change. For the following operations, each set has a peer responsible for all the sets, called *responsible*. In the case of Chord, the peer responsible is the one that manages the key 0. Each set has a identifier. In this description, for p sets, the identifiers will be $1..p$.

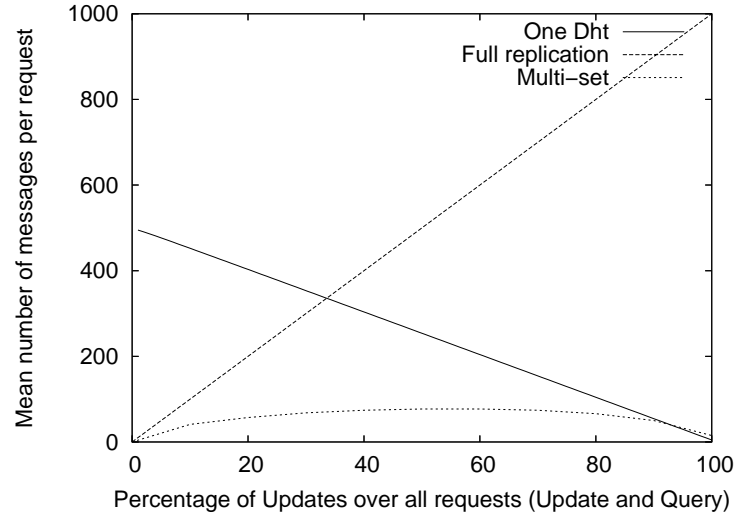


Figure 8: Model of the mean number of messages for a request depending on the ration Update/Requests. The two classical approaches are shown as a comparison. This system is composed of 1000 peers. p is chosen for each point of the Multi-set curve to minimize the mean cost.

3.4.1 Decision process

The right number of sets is decided on the responsible peers. Each responsible peers can initiate a change (increase or decrease) in the number of sets.

Responsible peers need to evaluate the current ratio of Update/Query in the system. To this end, all the peers send regularly the number and type of requests they received to their responsible. Each responsible uses those information sent by peers of its set and extract an evaluation of the ratio from them.

Using this estimated ratio and an estimation of the number of peers, the responsible uses the model to evaluate the optimal number of set according to the environment.

If a change is necessary, one responsible notices the others that a *Merge* or a *Split* is scheduled and initiate it.

Algorithm of *Decision process* on a responsible peer

```

clear(nb_query[], nb_update[])
time = get_current_time()
while(get_current_time() ≤ time + timeout)
    Recv(id, query, update)
    nb_query[id] = query
    nb_update[id] = update
end while

Request = Query = 0
forall(id)
    Request = Request + nb_query[id] + nb_update[id]
    Query = Query + nb_query[id]
end forall

optimal = model_get_nb_set(nb_peers, Query/Request)

if(optimal > nb_set) then split()
else if(optimal < nb_set) then merge()
end if

```

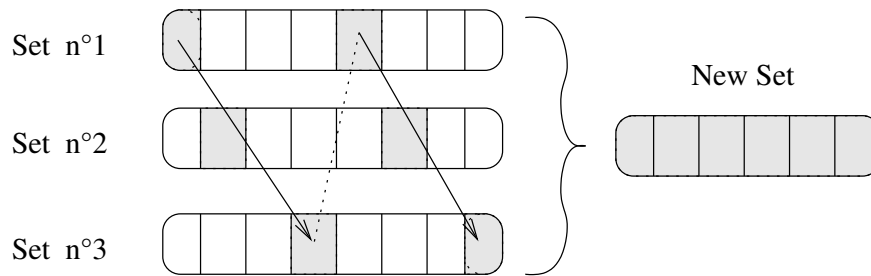


Figure 9: Choice of peers in a split operation. As keys and objects are replicated amongst adjacent peers, removing peers that are far from the others lead to reduce the number of messages during a split.

3.4.2 Merge

It is based on the capability of the underlying DHT systems to support peers that dynamically join/leave the system.

When a merge is decided, one of the sets is chosen to be destroyed. Then the peer responsible for this set ask the other responsible an estimation of their set size.

Then it evaluate how to use the peers of its set to eventually balance the number of peers in each sets.

Finally it send to each peers of its set the identifier of their new set. Then, they insert themselves in the other sets.

Algorithm of *Merge* on a responsible peer

```

clear nb_peers[]
forall(set except current_set)
    nb_peers[set] = request_nb_peers(set)
end forall

forall(peer in current_set)
    clear min
    forall(set except current_set)
        if(nb_peers[set] < min)
            then
                new_set = set
                min = nb_peers[set]
            end if
        end forall
        send(peer, "Change set to", new_set)
        nb_peers[new_set] ++
    end forall

```

After this operation is finished, the peers updates the links they had with the other sets are some are no more relevant.

3.4.3 Split

To be efficient, this operation is based on the error recovery mechanism of the underlying DHT system. In Peer to Peer systems like Chord, objects are replicated to prevent their lost when a client part unexpectedly from the system. For each peer, objects are replicated on its nearest neighbor. With such a system, when a peer leaves, the neighbor is naturally considered as the new owner of the key previously owned by the leaving peer.

The goal of the implementation of Split is to use these redundant information to prevent to do most of the communications.

Indeed, with taking neighbor in the old sets to create a new one, it would be necessary to: Communicate the lost information in the old sets to replace the lost information. Communicate in the new set to obtain the objects needed to complete the knowledge of the new set.

The first part can be reduced by choosing carefully the peer use to create the new set. Indeed by taking well distributed peers, it is possible to mostly remove peers whose objects are still replicated on other peers.

As shown on figure 9 we choose the peers the farther from one to another to create the new set.

Each responsible is assigned a number of peers to provide for the new set. It chose them in order to well spread them amongst its peers. The chosen peer then join the new set, keeping their old identifier in order to keep with them their knowledge. If there are gaps in this knowledge, they ask other sets using classical interval queries.

Algorithm of *Split* on a responsible peer

$step = nb_peers[current_set] / nb_requested_peers$

for $i = 1$ **to** $nb_peers[current_set]$

if $(i \bmod step = 0)$

then

 send(peer[i], "Change set to", new_set)

end if

end for

4 Performance evaluation

4.1 Methodology of evaluation

Multi-set is evaluated using a dedicated event-driven simulator implemented in Ocaml [11]. Each peers created the same number of requests by using the *generator of requests*. Requests stand for queries and updates. Queries are relative to intervals open on the right (like in *at least 32 free processors*) and with the left bound uniformly distributed over the keys space (see section 3.3). The underlying Peer to Peer architecture is Chord-like [13].

To evaluate such a system, two point of view are necessary. The user one, and the system one. Users are willing to obtain answer the fastest possible. The systems tends try not to consume too much resources and to prevent specific part of it to be overloaded. As the simulator is to be used for large systems, it simulates the network at high level, and thus the basic element of measure is the message. To accommodate the two points of view, the metrics are *Mean number of messages for requests (Query and Update)* and *Workload balance*.

Mean number of messages for requests gives a good indication of the latency for the user. Moreover the smallest it is the least global resources the system will need.

Workload balance is an indicator to verify if one peer does not answer all the requests. As this system aims to work efficiently without dedicated hardware, it is necessary to well balance the workload amongst all the peers.

Those values are measured by counting the number of messages for each request. Multi-set is compared with the two other limit approaches, i.e. a single DHT and total replication. Data for one DHT and total replication are obtained by simulation too. Models of the three approaches confirm simulation results.

4.2 Evaluation of the Query

First we evaluate the system behavior with respect to the system size. The following simulations are done with 4 sets.

Simulations are done to compare three approaches :

- One DHT
- 4-set
- 4-set when only the 10% of the total number of answers is retrieved

Figure 10 shows the mean number of messages needed to complete a Query for several number of peers for the three approaches.

More precisely, the 4-set mean value is the one of *One DHT* divided by the number of sets. Thus even if the cost remain linear, it is possible to reduce it by increasing the number of sets. Each time we duplicate the number of sets, the number of peers per set is instead halved, and thus also the mean number of messages needed to complete a query is halved too. Consider that this is because a single set of peers, hosting a replicated DHT network, is enough to solve an open interval query, and its cost is linear in the number of peers of the set.

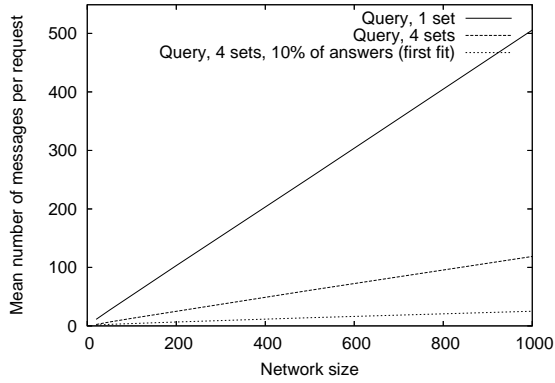


Figure 10: Simulation of the mean number of messages used to answer a Query for several approaches. The last one stop the query when 10% of the elements in the system are found.

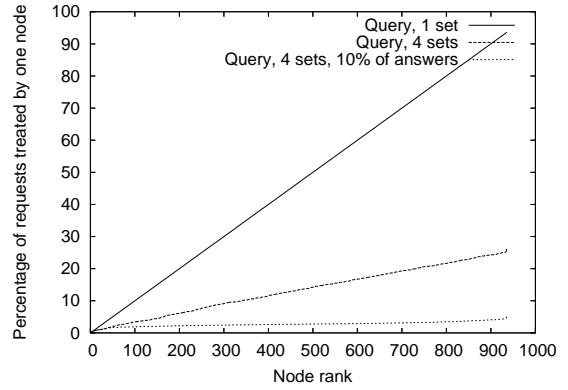


Figure 11: Simulation of the workload repartition for 1000 peers from the least loaded to the most one.

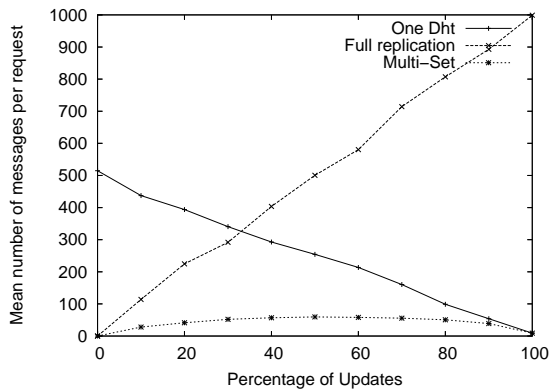


Figure 12: Simulation of mean number of messages in a 1000 peers system to answer Queries and Update. X-axis shows the percentages of the requests that are Updates.

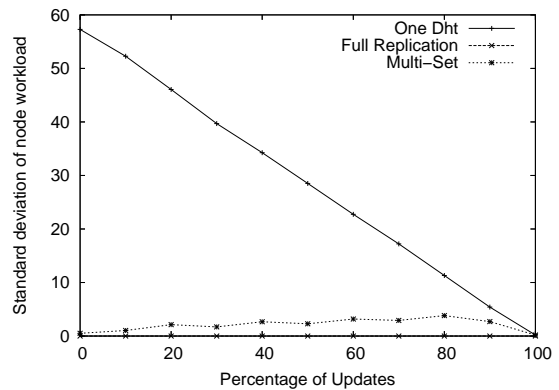


Figure 13: Standard deviation of the peer's workload in a 1000 peers system. X-axis shows the percentages of the requests that are Updates one.

The workload repartition follows the same trend. Figure 11 shows the workload repartition in a 1000 peers system for the three approaches. The worst cases workload for 4 sets are still the peers at the end of the intervals. But they just have to manage one fourth of the requests of the One-set case. For queries where only 10% of the answers must be retrieved, the load is well distributed. When the query has a very small first bound, most of the time it will stop far from the right end of the interval. Alas, this approach can't be used with multi-attribute queries as there are no guarantee that merging two partial results will give even one answer.

For Queries, rising the number of sets improves the mean cost as well as the workload repartition.

4.3 Dynamism of keys

The following simulation where done with different scenarios, from only Queries requests to only Update ones.

Figure 12 compares the performance of the three methods for a network of 1000 peers as a function of the ratio Updates/Requests. The Multi-set performance is always better than the others, particularly for non-extreme values of the ratio. The worst performance obtained for the Multi-set is when the ratio is 1/2. Yet, at this point it is more than 4 times more efficient than the system based on a single DHT, and 8 times more than the one based on total replication. This behavior confirms the model shown in part 3.3.

Figure 13 compares standard deviation of the workload assigned to each peer for the three methods for uniformly

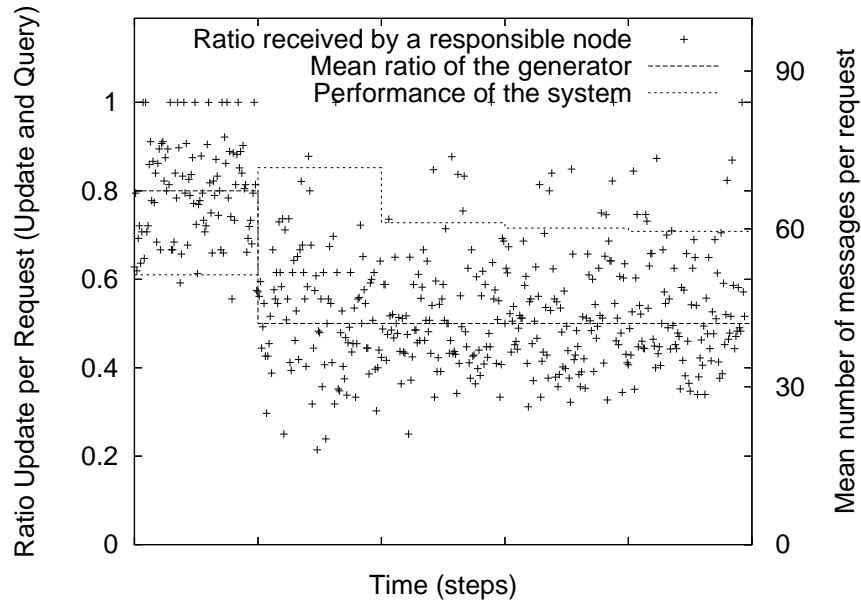


Figure 14: Simulation of the set number evolution according to the ratio in a 1000 peers system. The left Y-axis is the ratio of Updates compared to all the requests (Updates and Query). The right Y-axis is the performance of the system expressed as the mean number of messages per request.

distributed requests. Standard deviation of the total-replication is null as all the peers do always the same amount of work. Open queries for one DHT put a lot of weight on peers responsible for the end of the interval. By using Multi-set, the work is more distributed, even when there are mostly queries.

4.4 Dynamism of the ratio

Our approach is efficient because it does not need to have an a priori estimation for the number of sets. This number evolves in function of the ratio Updates/Requests by using the basic operations Merge and Split.

Figure 14 shows the use of the Split operation in a system of 1000 peers. At the beginning, the ratio is .8, then it changes to .5. At this point performance drops. After some data sent to the responsible peers, one of these decides to run a Split to improve the overall performances.

It is worth noting that the performance, expressed as the mean number of messages to answer a request, gets worse and reaches a peak when the ratio changes. This occurs until the Split operations ends up by including an optimal number of sets. In this case three Splits are done to obtain the optimal number of sets.

As shown on the model (figure 8) the system performance is the worst one when the ratio is 0.5. Thus, the performance decreases during the ratio changes from 0.8 to 0.5.

4.5 Experimental validation

Testing our system on a real platform would validate the previous simulations and model. Using a real production grid would limit the exploration of possible values for ratio and the nodes number. Moreover it would require a complete implementation of the Multi-set DHT, which would focus the performance evaluation on the underlying DHT rather than on the Multi-set to itself. For these reasons, a simpler implementation of Multi-set is evaluated. It runs on PlanetLab which provides a distributed infrastructure that is comparable to the infrastructure linking local Resource Managers in real Grids.

For these experiments, 60 PlanetLab computers emulate the whole infrastructure. To be able to experiment on larger scale, several peers run on each physical computer of PlanetLab. During experimentations, each peers on a

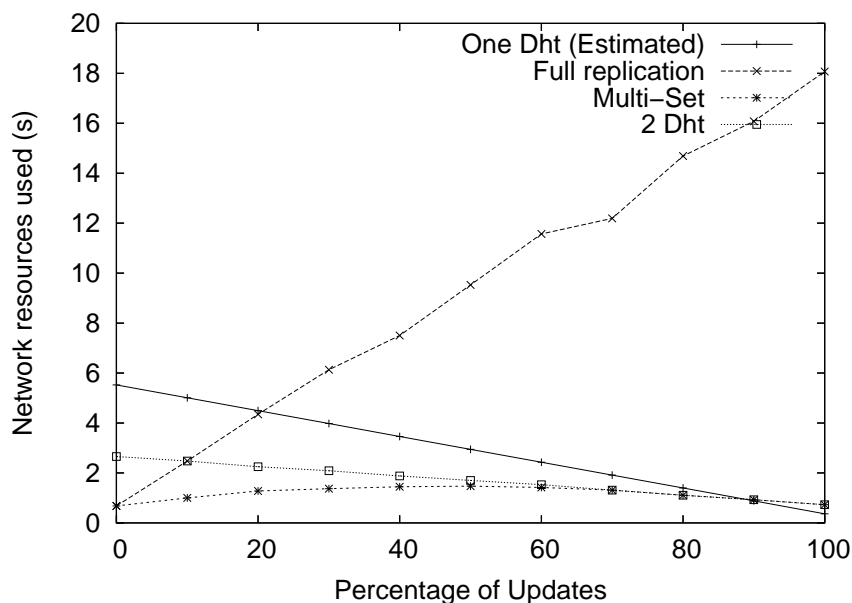


Figure 15: Experimentation on PlanetLab using 60 computers to emulate 120 nodes. Multi-set is used with between 2 and 20 sets.

computer belongs to a different set. This is done to prevent local communications to occur inside a single computer between peers. Such communications would be a bias as peers are supposed to be ran on different computers.

There are several available implementations of DHT [16, 13]. Most of them do not implement range requests and only manage classical Hash Table. A good and efficient implementation of classical DHT without interval queries, like Bamboo [16], is 20000 lines of Java and require heavy environments like a Java Virtual Machine (JVM) which becomes troublesome when running several peers on each computer.

We implemented a light interval enabled chord-like system in 1200 lines of python. Our implementation is aimed at providing basic interval enabled DHT system with low requirements (no JVM by instance) and ease of modification to simplify obtaining traces of execution. This naive implementation does not yet manage the dynamism of nodes by instance.

Each peer can reply to range query request, and can also generate requests, as in the previous simulation tests. Two metrics are used :

- Mean number of communication per request
- Total time of communication per request

During the 264 experiments, the first metric follows the model and simulation results. Total time of communication measures the global use of resources. Figure 15 shows that the experimental behavior roughly matches the simulations one. The main difference is that communications caused by range queries are more expensive than the one due to updates. In our experiment, range communication take 50% more time as they transport more information. This lead to a slight change of the optimal number of sets according to the ratio, but this can be included easily in the model by increasing the weight of requests. Comparing to the simulation and emulation results, full replication turns out to be less efficient as due to the broad distribution of latencies in PlanetLab.

5 Conclusion

Most of the Peer to Peer systems able to answer interval queries directly try to solve the problem by optimizing the Query and the Update costs. We choose to address a more realistic case where the ratio between the queries and updates is important. Our approach is to be used over other Peer to Peer systems able to deal with interval queries,

using them to manage each set. The resulting system performance is then dramatically better than the one underlying system used to manage each set. We can inherit some of their properties too, for example workload repartition.

The Multi-set method gives good results for each type of ratio Update/Request. Moreover it can adapt automatically to the way the requests are done. Thus it is efficient to manage several types of resources for Grids. This method is validated by experiments and simulations which confirm the proposed model.

The current model of workload is quite simple (uniform for the first bound of the queried interval) but most of the underlying Peer to Peer systems use workload repartition algorithms.

References

- [1] *EGEE User's Guide*, May 2006.
- [2] Ian Foster, Adriana Iamnitchi. *A Peer-to-Peer Approach to Resource Location in Grid Environments*. In Symp. on High Performance Distributed Computing, 2002.
- [3] Nicolas Capit, Georges Da Costa, Yiannis Georgiou, Guillaume Huard, Cyrille Marti n, Grgory Mouni, Pierre Neyron, and Olivier Richard. *A batch scheduler with high level components*. In Cluster computing and Grid 2005 (CCGrid05), 2005.
- [4] Ben Clifford. *Globus monitoring and discovery*. In GlobusWorld05, 2005.
- [5] David Patterson David Oppenheimer, Jeannie Albrecht and Amin Vahdat. *Scalable wide-area resource discovery*. Technical Report UCB/CSD-04-1334, EECS Department, University of California, Berkeley, 2004.
- [6] Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. *Online balancing of range-partitioned data with applications to peer-to-peer systems*. Technical report, Stanford U., 2004.
- [7] Steven D. Gribble, Alon Y. Halevy, Zachary G. Ives, Maya Rodrig, and Dan Suciu. *What can database do for peer-to-peer?* In Fourth International Workshop on the Web and Databases (WebDB '2001), pages 31-336, 2001.
- [8] H. V. Jagadish, Beng Chin Ooi, and Quang Hieu Vu. *Baton: a balanced tree structure for peer-to-peer networks*. In VLDB '05: Proceedings of the 31st international conference on Very large data bases, pages 661-672. VLDB Endowment, 2005.
- [9] Nikos Ntarmos, Theoni Pitoura, and Peter Triantafillou. *Range query optimization leveraging peer heterogeneity*. In 3rd International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P 2005), August 2005.
- [10] Andy Oram. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
- [11] Didier Rmy. *Using, Understanding, and Unraveling the OCaml Language*. From Practice to Theory and Vice Versa, chapter 2002, pages 413-536. Lecture Notes in Computer Science.
- [12] Ozgur D. Sahin, S. Antony, Divyakant Agrawal, and Amr El Abbadi. *Probe: Multi-dimensional range queries in p2p networks*. In WISE, pages 332-346, 2005.
- [13] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. *Chord: A scalable peer-to-peer lookup service for internet applications*. Technical Report TR-819, MIT, March 2001.
- [14] Paolo Trunfio, Domenico Talia, Paraskevi Fragopoulou, Charis Papadakis, Matteo Mordacchini, Mika Pennanen, Konstantin Popov, Vladimir Vlassov, and Seif Haridi. *Peer-to-peer models for resource discovery on grids*. In Proc. of the 2nd CoreGRID Workshop on Grid and Peer to Peer Systems Architecture, 2006.
- [15] David R. Karger, Matthias Ruhl. *Simple Efficient Load Balancing Algorithms for Peer-to-Peer Systems*. In SPAA '04: Proceedings of the sixteenth annual ACM Symposium on Parallelism in Algorithms and Architectures, 2004.
- [16] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiataowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. *OpenDHT: A Public DHT Service and Its Uses*. Proceedings of ACM SIGCOMM 2005, August 2005.