Multi-set DHT for range queries on dynamic data for Grid information service

Georges Da Costa[1], Salvatore Orlando[2], and Marios D. Dikaiakos[3]

[1] IRIT - Universite Paul Sabatier, Toulouse III, France
[2] University of Venice, Italy
[3] Department of Computer Science, University of Cyprus, 1678 Nicosia, Cyprus

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. 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 indexed data. Evaluation is done by modeling, simulation and experimentation on PlanetLab. The use case is an information service for Grid, where resource attributes are indexed.

1 Introduction

Grids are based on several basic but nevertheless necessary services. One of such services is the information service which manage resources. This service has to keep track of the Grid state and to locate resources corresponding to user queries. 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.

Information service have thus to track the various Grid elements and characteristics. To this end, they have to manage large amounts of distributed and changing information. Information service have to answer to queries which are initiated by users willing to run their applications on the Grid. Moreover, such a system should be able to efficiently answer queries [12] such as: *find clusters with at least 32 free processors and ATM network*.

Difficulties arise due to the large scale of current Grids. 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.[2].

Peer to Peer systems are known [9] for having solved the file-sharing scalability problem and for being fault-tolerant and easy to deploy and manage. However, classical Peer to

[3] This work was funded by Ercim at CNR-ISTI (Pisa, Italy) and at UCY (Nicosia, Cyprus). This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

Peer systems (Chord, Freenet) cannot directly be used to solve the problem of Grid information service, 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 information services have been already introduced [1, 4]. Even links between databases and Peer to Peer have been explored [6]. 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 [1, 4] using a Peer to Peer approach to provide dynamism aware information service. Users ask about particular resources such as: *Where can I find computers with ATM network*, or range of resources such as: *Where can I find storage with at least 1Tb 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[4] into logical expression of simple queries about particular resources or range ones. The Grid information service 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) into this model. Thus a Grid information service can be built using a simple module that answers range 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 Related works on range queries

2.1 Context

In the following we will concentrate on the problem of solving range queries and updates on a single resource attribute. The way of managing more complex queries using such basic component can be found in [4].

To this end, we consider 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 a range 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.

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.

Such system needs to react fast to changes to prevent from giving false information 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.

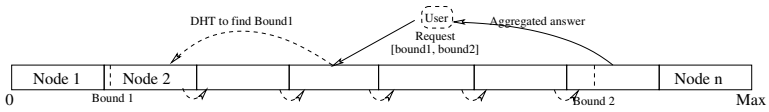

Fig. 1. Key space is split in ranges, one for each peers. To find a range, one has to search for the peer responsible for the range's lower bound, using the global DHT. Then this peer contacts its neighbor, and so on until the upper bound of the range is reached.

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: locate the peer which could answer a query on a particular key. Typically, the worst case cost to locate an object, in terms of number of messages, is logarithmic with the number of peers. For example, Chord [11] guarantees queries need less than $log(participants)$ messages. 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 range query problem: for instance Probe [10], Baton [7], and others [8, 5] described in [12]. These systems use the range technique (Figure 1) to provide range functionality.

The query cost of those systems is linear in the size of the range. Keys are spread amongst all the peers, so the larger the range is, the more peers it involves. The second query 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* range queries, the peer responsible for one of the two extremities of the key space has to answer all the queries. Using one DHT appears to be efficient only when most of requests are updates.

2.3 Other methods

Another extreme method would be to maintain a total replication of all the objects to be indexed amongst the peers. Queries are free in terms of messages exchanged as they can be processed locally. But changes of resource attributes become very expensive since all peers must be contacted to update their index. This method is suitable when queries largely outnumber updates as cost for updates is linear, and query cost is constant.

3 Mutli-set approach

In this section we discuss our proposal to manage objects of dynamic characteristics. In the case of a Grid information service, the number of free processors in each site is dynamic. In this example, *object* is the site URI[4], and *characteristic* is the number of free processors of the site.

[4] *Universal Resource Identifier*, used to contact a site

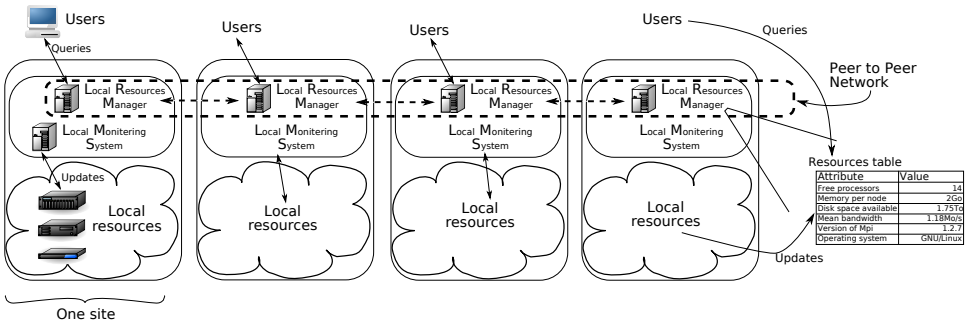

Fig. 2. Each site uses at least one Local Resource Manager (such as GRAM). This *lrm* receives updates concerning the resources it manages locally. It receives queries from users too. When put together in a Peer to Peer network, they forward queries and updates according to local algorithms.

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 peers of a Grid system, and the case where the resource index is fully replicated to all Grid peers.

Grid information service is based on information made available by each resource. In the following we consider that *peer*s are the computers that manage the local resources of the Grid sites. Usually, each cluster of the Grid uses a local resource manager such as GRAM in Globus. This allows us to obtain a fully decentralized system. Each user is connected to one of these peers. As a simplification we will only consider the site, the user, and the peer he uses as a single entity. A diagram of this structure is shown in Figure 2.

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 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 *set*s are approximately of the same number of peers.

3.1 Sets

Sets are disjoints groups of peers (Figure 3). Sets are a partition of all the peers. Each of these sets behaves like an range-enabled object management system. All these sets manage the same information. Thus all pairs (key, objects) , where key is ¡attribute, value¿, and objects are the list of all Grid resources r which r.attribute = value holds, are duplicated on each set. With this property, a query returns the same value, independently of the chosen set. In order to synchronize the sets, updates are 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

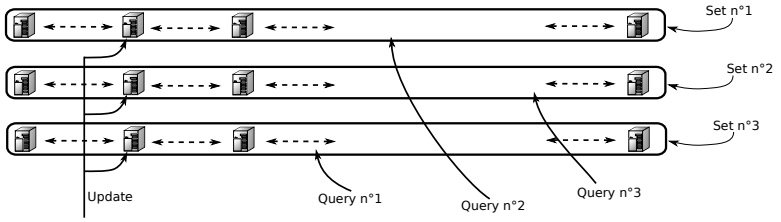

Fig. 3. Multi-set system: Objects are replicated across several groups of peers called *set*s. An *Update* is forwarded to all the sets. A *Query* goes to a random set. The number of sets depends on the ratio Query/Update

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, sets are implemented as range enabled DHT systems.

The two fundamental operations provided (Figure 3) 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.

3.2 Trade-off

Multi-set DHT with one set corresponds to *a single DHT* case, whereas one peer per set corresponds to the total replication case.

The Multi-set approach lies between those two extremes. As the ratio Update/Requests evolves over the time, it adapts *update* and *query* costs depending on its use. With this system, *Updates* become more costly as the number of sets increases, but *Queries* become less expensive. By adapting the number of sets to the ratio Query/Update, a good trade-off can be achieved. Coordinators obtain information from peers to deduce 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, metrics 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 latency and on resources used by the system. As the underlying Peer to Peer systems can adapt to spread requests uniformly amongst the peers [13], 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 range queries open on right, and the left bound will be uniformly distributed.

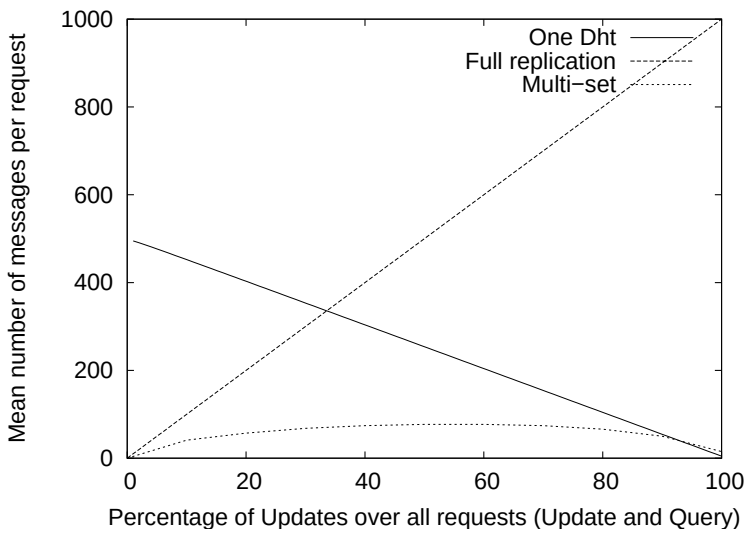

Fig. 4. Model of the mean number of messages for a request depending on the ration Update/Requests. The two classical approaches are shown as a comparison. This system is composed of 1000 peers. p is chosen for each point of the Multi-set curve to minimize the mean cost.

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 [11].

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 range. The mean size of the range 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 No communication as all objects are stored locally.

Query cost for total replication All peers are updated: $N - 1$ communications

Update cost for p sets 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: $2p \ln(\frac{N}{p}) + (p - 1)$

Query cost for p sets There is only one set that answers the query. 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: $\alpha * (2p \ln(\frac{N}{p}) + (p - 1)) + (1 - \alpha)(\frac{p-1}{p} + \frac{N}{2p})$

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

The curves on figure 4 show the three costs. For each α, the p used for the Multi-set is the one that minimizes the mean cost.

3.4 Algorithms

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

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

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 peers know at least one peer of each other sets. Decisions do not occur frequently beyond initialization as patterns of use do not often change. For the following operations, each set has a peer responsible for all the set, called *supervisor*. In the case of Chord, the supervisor is the one that manages key 0. Each set has a identifier. In this description, for p sets, the identifiers will be $1..p$. Pseudo-code for those algorithms can be found in [3].

Decision process The right number of sets is decided by the supervisor. Each supervisor can initiate a change (increase or decrease) in the number of sets.

Supervisors need to evaluate the current ratio of Update/Request in the system. To this end, all peers send regularly the number and type of requests they received to their supervisor. Each supervisor uses those information sent by peers of its set and extracts an evaluation of the ratio from them. Using this estimated ratio and an estimation of the number of peers, the supervisor uses the model to evaluate the optimal number of set according to the environment. If a change is necessary, one supervisor notifies the others that a *Merge* or a *Split* is scheduled and initiates it.

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 set is chosen to be destroyed. Then the supervisor for this set asks the other supervisors an estimation of their set size. Then it evaluates how to use the peers of its set to eventually balance the number of peers in each sets. Finally it sends to each peers of its set the identifier of their new set. Then, they insert themselves in the other sets. After this operation is finished, peers updates the links they had to other sets as some are no more relevant.

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, indexed data are replicated to prevent missing them when a client depart unexpectedly from the system. For each peer, objects are replicated on its nearest neighbor. When a peer leaves, its 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 most communications.

Each supervisor is assigned a number of peers to provide for the new set. Thus the new set will be composed of some peers of all sets. It chooses 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 range queries.

4 Performance evaluation

4.1 Methodology of evaluation

Multi-set is evaluated using a dedicated event-driven simulator implemented in Ocaml. Each peers created the same number of requests by using the *generator of requests*. Requests stand for queries and updates. Queries are relative to ranges 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 [11].

To evaluate such a system, two points of view are necessary. The user one, and the system one. Users are willing to obtain the fastest possible answer. The system tries 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 performance measure (e.g., to evaluate query resolution) is the number of messages exchanged. 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 latency for users. Moreover the the less the exchanged messages are, the less the amount of resources are needed.

Workload balance is an indicator to verify if unpleasant situations occur, as a single peer that answers all the requests. As this system aims to work efficiently without dedicated hardware, it is necessary to well balance workload amongst 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. Simulation results confirm the models of the three approaches.

Simulation and model are shown for 1000 nodes, experimentation for 120 nodes. Such large values are chosen to evaluate the scalability of the approach for future large Grids.

4.2 Dynamism of keys

The following simulations where done with different scenarios, the limit case with only Queries requests to the other limit case with only Updates.

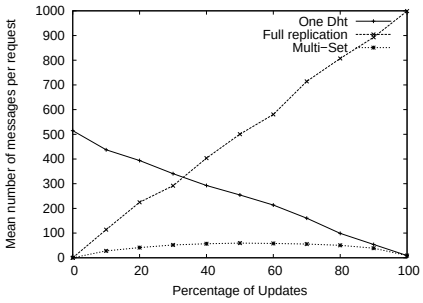

Fig. 5. Simulation of mean number of messages in a 1000 peers system to answer Queries and Update. X-axis shows the percentages of the requests that are Updates.

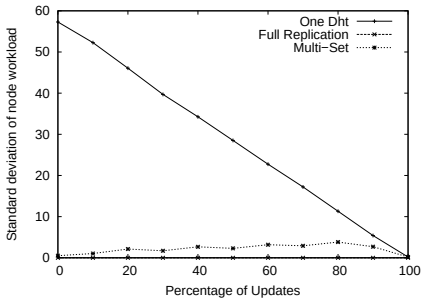

Fig. 6. Standard deviation of the peer's workload in a 1000 peers system. X-axis shows the percentages of the requests that are Updates one.

Figure 5 compares performances of the three methods for a network of 1000 peers as a function of the ratio Updates/Requests. 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 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 Section 3.3.

Figure 6 compares standard deviation of the workload assigned to each peer for the three methods for uniformly 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 range. By using Multi-set, work is more distributed, even when there are mostly queries.

4.3 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 7 shows the use of the Split operation in a system of 1000 peers. At the beginning, ratio is $.8$, then it changes to $.5$. At this point performance drops. After some data sent to the supervisor, 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 4) 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.

4.4 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

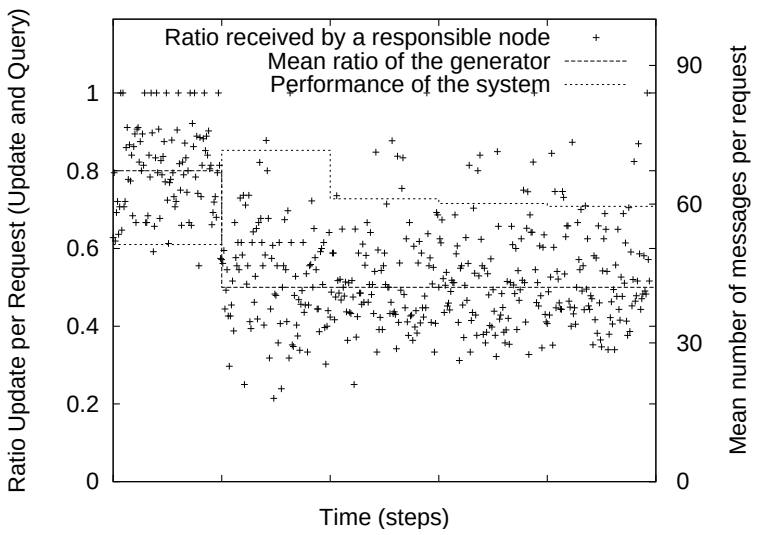

Fig. 7. Simulation of the set number evolution according to the ratio Updates/Requests (1000 peers). Left Y-axis is the ratio (dots and large dashes). Dots are the ratio values sent to supervisors. Large dashes is the mean value of these ratio. Right Y-axis (light dashes) is the performance (mean number of messages per request). Performance decrease when ratio changes, then the system adapts the set number to improve performance.

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 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 to real Grids. In this experimentation, the metric is the total time of execution. Compared to the previous simulations which only allow us to measure the number of exchanged messages, the results obtained by exploiting PlanetLab are more interesting. For example, they take into account also the real contents and size of messages.

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

There are several available implementations of DHT[14, 11]. Most of them do not implement range requests and only manage classical Hash Table. A good and efficient implementation of classical DHT without range queries, like Bamboo[14], 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.

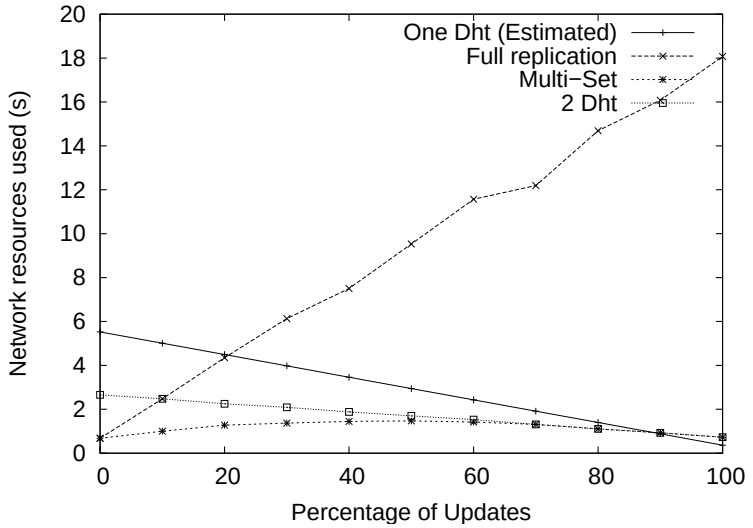

Fig. 8. Experimentation on PlanetLab using 60 computers to emulate 120 nodes. Multi-set is used with the number of sets ranging from 2 to 20 sets.

We implemented a light range enabled chord-like system in 1200 lines of python. Our implementation is aimed at providing basic range 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 requests, and can also generate requests, as in the previous simulation tests. Two metrics are used : *Mean number of communication per request* and *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 8 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 queries. In comparison to the simulation and emulation results, full replication turns out to be less efficient, and this is due to the broad latencies distribution in PlanetLab.

5 Conclusion

Most of the Peer to Peer systems able to answer range 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 can be used to improve the perfomance of a P2P system able to answer range queries. In particular, since our proposal requires to partition all the participating nodes in several

sets, the same P2P network organization can be exploited within each set. The resulting system performance is then dramatically better than using a global P2P system spanning all the peers. We can inherit some of their properties too, for example workload repartition between peers.

The Multi-set method gives good results for each type of ratio Update/Request. Moreover it can adapt automatically when this ratio change over time. Thus this can efficiently used to manage several types of attributes associated with different Grid resources. 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 range query). However most of the underlying Peer to Peer systems are able to modify the assignment of keys to accommodate fair workload distribution.

References

1. Ian Foster, Adriana Iamnitchi. *A Peer-to-Peer Approach to Resource Location in Grid Environments*. In Symp. on High Performance Distributed Computing, 2002.
2. Ben Clifford. *Globus monitoring and discovery*. In GlobusWorld05, 2005.
3. Georges Da Costa, Salvatore Orlando, Marios D. Dikaiakos. *Multi-set DHT for interval queries on dynamic data*. Coregrid Technical report TR-0084, 2007.
4. 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.
5. 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.
6. 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.
7. 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.
8. 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.
9. Andy Oram. *Peer-to-Peer : Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.
10. 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.
11. 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.
12. 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.
13. 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.
14. Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. *OpenDHT: A Public DHT Service and Its Uses*. Proceedings of ACM SIGCOMM 2005, August 2005.