On Overlay Schemes to Support Point-in-Range Queries for Scalable Grid Resource Discovery

Liping Chen K. Selçuk Candan Junichi Tatemura Divyakant Agrawal
Dirceu Cavendish

NEC Laboratories America, Inc.
10080 North Wolfe Road, Suite SW3-350, Cupertino, CA 95014, USA
E-mail: {liping,candan,tatemura,agrawal,dirceu}@sv.nec-labs.com

Abstract

A resource directory is a critical component of a Grid architecture. P2P computing paradigm could address some of the scalability issues that make distributed resource discovery services challenging. Unfortunately, most existing Distributed Hash Table (DHT) based P2P overlays have difficulty in treating attribute range queries that are common in resource discovery lookups. This paper proposes a general framework for range-based resource discovery. In particular, the proposed framework maps tree-structured logical data (i.e., range indexing) onto a DHT-based physical node space (i.e., resource brokers). In this paper, we consider three mapping schemes from the logical space onto the physical space. Each mapping scheme uses a different replication mechanism to reduce range search time and to achieve load balance. We analytically and experimentally compare the performance characteristics (query/update costs and workload distributions) of these schemes and discuss their applicability under different resource discovery service scenarios.

1. Introduction

A critical component of a Grid infrastructure is the *information service* (or *directory service*) component, used by the Grid clients to locate the resources over the network. The three currently implemented information service components are the Globus Toolkit Monitoring and Discovery Service (MDS2,3,4), the European Data Grid Relational Grid Monitoring Architecture (R-GMA), and Hawkeye, part of the Condor project. All of these three discovery services are based on a centralized architectural design. Each of them works reasonably well for current Grid platforms for highly specialized interests groups (e.g., multiple workstation clusters being shared by a small group of material scientists). Zhang et al. [19] recently conducted a performance study of the above three systems and have experimentally verified that all three monitoring and discovery systems fail to scale beyond 300 concurrent users; i.e., the throughput begins to decline below acceptable levels. With respect to response time, MDS2 performs the worst, R-GMA is in the middle, and Hawkeye is the best among the three. The study indicates that unless the scalability problem of monitoring and discovery services is resolved, commercializing Grid computing will be successful only in limited application domains. In this paper, we note that, P2P computing paradigm could address some of the scalability issues that make distributed resource discovery services challenging. In this process, we investigate and compare different strategies for point-in-range search queries in P2P systems.

1.1. Problem Statement

In the context of grid computing architectures, there are three main components: resources and resource providers, users, and brokering (or match-making) service which acts as an intermediary between the users and the resources. In this paper, we focus on the brokering service. In the case of Globus [8], the brokering service is implemented as a centralized architecture (MDS2). A brokering service based on the P2P paradigm would instead create a distributed hash table (DHT) based overlay of broker nodes, which collectively maintain the index information to facilitate resource discovery in computational grids. The major challenge in designing such an overlay is that in contrast to other P2P data sharing environments, access to resources in the grid domain is based on attribute values of the resources and not on names. Thus, P2P systems, such as Gnutella or Napster, cannot be used directly since these systems support name-based lookup. In this paper, we adopt a simple, but general approach:

each resource is represented by a (K, L) pair. Each key K is a vector of attribute values to specify a resource and L is the address of the resource. For example, a key to describe a Linux server resource could be represented by a vector: (version, CPU, memory, permanent storage, amount). Note that the amount of a resource available at a provider may change over time and may therefore be categorized as a dynamic attribute.

Central to any directory service is an indexing scheme which maps keys to resources. One source of complexity that arises in Grid resource discovery is that the lookups are not point lookups (or exact lookups) but range lookups. For example, a user request for a 500MHz CPU can also be satisfied with a CPU of 800MHz. None of the pure DHT-based schemes are capable of providing such range-based lookup functionality. An obvious problem with the use of a DHT overlay in range search is the inherent randomness of the hashing function: efficiency in range searches requires pruning of the search space and this can be done only if the entries are stored in a way that makes their logical relationships explicit; randomness inherent in DHT schemes, on the other hand, prevents enforcement of any such explicit order.

Recently, there have been various attempts to solve the range search problem over DHT networks [1, 10, 15, 18]. Many of these schemes are designed with the goal of supporting database type applications, where usually the user needs to receive **all** data points in a given query range. However, we note that in the context of resource brokering, we are only interested in any one point (a single physical resource) which lies within the query range (resource attribute constraints). Therefore, unlike traditional range queries, for grid resource discovery, it is enough for the system to identify **any one** point (i.e., resource) within the given range. We refer to this type of queries as *point-in-range* (or p-range) queries. Existence of p-range queries has impact on how the distributed directory is structured and how queries are processed.

1.2. Contributions

In this paper, we consider three schemes for mapping a *resource directory*, represented as a tree-structured logical data space to answer p-range queries, onto a DHT-based physical node space:

- The tree replication scheme (TRS), discussed in Section 4, replicates the entire tree data structure.
- The path caching scheme (PCS), introduced in Section 5, replicates paths from root to leaves
- The node replication scheme (NRS). presented in Section 6, replicates individual logical nodes.

Once an appropriate mapping scheme is chosen, any DHT scheme can be used to route requests. Therefore, instead of proposing a new tree structure or a new DHT overlay, in this paper we focus on investigating the properties of these three general mapping schemes, usable with different tree structures and DHT overlays. We note that the workings of the underlying replication scheme impacts the overall effectiveness of the resource brokering service. Therefore, we analytically and experimentally compare and contrast these three schemes under different scenarios and describe their performance characteristics, in terms of *lookup cost*, *update cost*, and *redirection workload* per peer.

2. Related Work

Foster and Iamnitchi [7] have articulated the eventuality of the convergence of Grid computing and P2P computing paradigms. Naturally, an early proposal to use the P2P paradigm to support resource discovery is due to them [13]. The connection between Grid Information Services and range query processing over DHT-based P2P systems has been made by Adrzezak and Xu [2]. They propose a solution based on CAN [16] and use space-filling curves to perform range query processing in P2P environments.

The notion of range query processing over data distributed over P2P systems has since then been identified as an important problem by several research groups [11, 3, 12, 14, 9, 6, 5, 1, 10, 15]. Gupta et al. [11] have developed a locality-sensitive hashing based mechanism for approximate range query processing over P2P networks. However, the environment they consider is one in which peers cache the results of prior answers to range queries; the data itself is not distributed or partitioned over the peers. Aspnes and Shah [3] have developed a generalization of skip lists referred to as skip graphs over distributed systems to support range query functionality over data distributed over P2P systems. Skipnet [12] has also been developed independently to skip graphs, which is essentially identical to the latter. Skipnet can also be extended to process range queries over P2P systems. Sahin et al. [14] have developed an exact range query processing architecture by utilizing an underlying CAN [16] based overlay over P2P systems. Here too the authors assume that the data is not range partitioned, but rather the peers cache the results of prior queries which can be used to answer future queries instead of processing the query directly at the data source. More recently, database researchers have begun investigating range query support over P2P systems. In particular, Ganesan et al. [9] propose a load balancing approach to partition the data ranges over P2P systems so that there is an absolute bound on the load skew among peers with respect to the storage. The underlying range query lookup mechanism is implicitly assumed to be either skip graph or skipnet. Crainiceanu et al. [6, 5] have recently proposed a generalization of B-tree like balanced search structure for processing range queries over P2P systems. P-grid [1] is a fully decentralized randomized protocol which ensures that the number of hops along the logical path is bounded by *log* of the number of nodes. [10] proposes a scheme where all peers know the entire logical structure, therefore avoiding traversals during lookups. We note that, all these schemes are designed and analyzed for range queries, which return all points in a given range. Instead, in this paper, we focus on the point-in range (p-range) queries.

3. Distributed p-Range Indexing for Grid Resource Discovery

Central to the efficiency of the directory for p-range lookups[1] is a scheme to map the logical data structure on the physical space covered by the DHT overlays.

3.1. Logical Space

There are many tree structures (e.g., B-tree for 1-D data and k-d tree, R-tree, etc. for multidimensional data) that support efficient range searches. These differ in the ways they split the space for efficient access and the ways they manage the corresponding data structure (e.g., balanced vs. un-balanced). Most balanced index structures provide $O(logN)$ point- and $O(N)$ range-search time (where N is the number of nodes in the tree); but, updating these index structures is costly since balancing may require restructuring the entire tree. Unbalanced index structures do not have restructuring costs (updates can be performed locally), yet in the worst case, they can lead to $O(N)$ search time, even for point queries.

In this paper, without any loss of generality, we use a simple binary space partitioning to maintain the 1-D data[2]. A suitable partition naming scheme is bit interleaving: starting from the root, we assign 0 to the left branch and 1 to the right branch [1, 15]. Furthermore, the range associated with each branch is determined by the value chosen for the split. Thus, for example the split at value $x1$ specifies that all queries with $x < x1$ should be directed to the left branch and others with $x > x1$ to the right branch. Splitting performed until the desired occupancy level is reached for every partition. Therefore, each leaf node (partition) has a unique name, which can be mapped to peers.

3.2. Physical Space

Physical nodes in the network correspond to the resources and brokers in the system. In general, assuming that brokers are responsible by similar number of resources, the number of nodes in the logical and physical spaces are linearly correlated. The physical space is a Distributed Hash Table (DHT) based structured overlay. The basic idea of DHT is to map keys to physical nodes using a consistent hashing function, usually SHA-1. In this paper, we use Chord [17] as the physical network; however, *any DHT implementation would work.* Note that recent results [4] show that, despite what is commonly believed, exploiting heterogeneity and supporting complex queries are not fundamental problems of structured overlays. In fact [4] presents efficient flooding and random walks on structured overlays and shows that structural constraints achieve low maintenance overhead. Therefore, in this paper, we limit our discussion to structured DHT overlays (however, we highlight where flooding might be more effective then structured maintenance).

Chord orders the identifiers in a ring modulo 2^m, where 2^m is the size of the identifier space. Both the key space and the node address space (i.e., IP addresses) are mapped to this same identifier space. A given key, k, is assigned to the first node whose identifier is equal to or immediately follows k in this circular space. Chord ensures that for a network overlay of N nodes the lookup cost is $O(logN)$ [17].

3.3. Mapping the Logical Space onto the Physical Space

Several factors play important roles in choosing a P2P paradigm for implementing resource discovery: *scalability*, *load balancing*, and *self-reorganization*. Inherent to any scalable mapping of the tree-based logical space to the physical space is the need for *replication* to ensure that higher level tree nodes in the tree are stored redundantly in the DHT overlay so that we can achieve both *scalability* (in terms of preventing full root-to-leaf traversals) and load-balancing.

Furthermore, the mapping should be sufficiently flexible so that we can still preserve the property of self-organization, in that the system allows dynamic inclusion (peer join) and exclusion (peer departure/failures)

1 P-range queries were introduced in Section 1.1.
2 However, the mapping techniques would equally apply to other 1-D and n-D tree structures as well.

of the physical nodes. Whenever a leaf node is overloaded due to the skewed distribution of data points, a *split* operation is incurred to split the leaf node. The split operation introduces a transient phase into the network. This happens when a partition L has been re-partitioned into two new partitions, L_1 and L_2, but this re-partitioning has not been reported to all relevant peers. During this period, L has to redirect any queries that wrongly targets at L to either one of the two new partitions L_1 and L_2. Overall, the split cost comes from two different aspects of the network: (a) required maintenance of the data structure due to the repartitioning of the node into smaller ranged nodes and (b) propagation cost to reflect the updates to all replicas within the entire network.

In this paper, we present three protocols for mapping tree-structured logical data onto a physical space, where, given the *logical id* of a *logical data node* as the key, a DHT-based structured overlay is used to locate a *physical node* which contains the *logical node* being searched. These three schemes (tree replication (TRS), path caching (PCS), and node replication(NCS)) differ in the granularity with which they use *replication* to help search efficiency and achieve load balance. In Section 7, we evaluate costs and benefits of the proposed replication schemes.

Note that, in the context of search structures, there are two types of information that needs to be maintained: (1) the search hierarchy (pointers etc.) used for navigation through the search space and (2) the actual data which corresponds to the detailed information about the attributes as well as provider's *address* of a specific resource. Different mapping schemes treat these differently. For example, [1] stores the actual data only at the leaves and implements a search hierarchy using available peers. Range Search Tree (RST) [10], on the other hand, stores data in its internal tree nodes, yet assumes that the logical structure corresponding to the search tree is known.

4. Tree Replication Schemes

The tree replication scheme (TRS) replicates the search hierarchy in its *entirety*; i.e., certain nodes in the physical space contain replicas of the entire index structure. In this respect, this is similar to the scheme use in RST [10]. The difference is that RST aims to return all results in a range, while resource discovery queries (i.e., $\exists$ range queries) require any match within the range. Any search operation has to first reach one of these nodes to access the index and find which physical nodes contain the leaves corresponding to the requested range. Replication of the index structure is used for improving scalability: a centralized scheme with only one index node would hardly be scalable [19].

Replication. Unfortunately, a simple analysis shows that to achieve load scalability, the number of directory replicas should be $O(N)$, where N is the total number of nodes in the network. Assume that on average each node receives c request/sec from users, where c is a constant. The total directory look-up load per second is therefore $L = cN$. If there are K replicas, the load distribution would be $O(N/K) = O(N)$ on each copy. This simply means that each physical node should be aware of the entire tree to have a constant directory lookup load. This is consistent with [10].

Lookups. If each node knows the entire tree structure, the query look-up would be cheap for this scheme: DHT lookups are needed simply to locate matched labels. Since each DHT lookup costs $O(logN)$ access, the total look-up cost of the tree replication scheme is simply $O(logN)$. In general, if there are K replicas, the search would require $O(logN)$ time to locate one of the K index nodes and $O(logN)$ time to locate one of the matches. Thus the search requires $O(logN)$ time.

Updates. In general, if there are K replicas of the directory, each update of the directory structure requires $O(KlogN)$ messages to update these replicas. Since in a resource brokerage network available resources may change frequently, updates to the index structure are not rare; thus, the TRS may be overly expensive. Note that since the search hierarchy is never traversed, height balancing is not needed.

Workload. In this scheme, the directory lookup load is shared equally among the K replias, while the DHT redirection load is shared equally among the N peers.

Summary. Although it has a good lookup behavior and load balancing characteristcs, this approach suffers from high update cost and large directory sizes. Note that, in many tree index structures, lower nodes split more frequently. Reducing the amount of lower node replication is thus expected to reduce the update cost. The number of replicas of each node should be proportional to the number of its leaf descendants. The next two schemes are based on this observation.

5. Path Caching Schemes

Most prefix-based range search structures (such as P-grid [1] or [15]) are based on path caching schemes, which construct a single logical tree and perform replication at the physical level. Each physical node has only a carefully selected partial view (the path from the root) of the logical tree. We use a simple example to illustrate how path caching works. Suppose we construct a simple logical tree as shown in Figure 1:

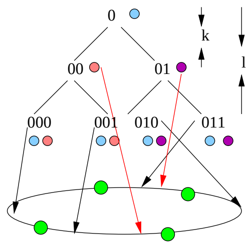

Figure 1. Path caching scheme

- Each tree node is named by the naming scheme we introduced in Section 3.1.
- The leaf nodes are mapped to physical nodes.
- Internal nodes are also mapped to physical nodes. Furthermore, the content of the internal nodes are replicated by *all* its leaf descendants.

Hence the root is replicated everywhere in the physical space; whereas the content of tree node 00 is replicated only at physical nodes which store logical nodes 000 or 001. Another way to look at it is that each physical node stores the information about the *entire path* from the root to the leaf node that is mapped to it (hence the name path caching scheme). Suppose a query originates from tree node 000 (the lower left tree node) and the target leaf node is 011 (the right most node). Since node 000 also stores (or caches) the contents of the tree node 00 and 0, it knows immediately that the request should be routed through a node containing 01.

Lookups. Note that, in the above example, *the node where 01 is explicitly mapped is our subtarget for routing, but it does not need to be reached every time: a replica of 01 might be found on the way.* Thus, its replicas will take part of the load of 01 away and reduce the number of hops a request need to travel. Of course, the load balancing efficiency of the scheme depends on the probability with which replicas are hit before reaching the target. Suppose a tree is balanced with height $h = logN$, and the level of the target node is k (i.e., the target node has 2^{h-k} replicas). The probability that one of the replicas will be hit before the target is hit (during $logN = h$ steps of DHT routing) is $1-(1-2^{-k})^h$. Therefore, it is not very likely that a search will need to go through all levels bottom-up and then top-down. In fact, [1] shows that their randomization scheme can ensure that the *expected* number of hops along the logical path needed is bounded by $log(N)$. Thus the search cost is $O(logN \times logN) = O(log^2N)$. In general, in a non-height-balanced tree, the *worst case* search cost could be upto $O(h \times logN)$ where $1 \leq h \leq N$ is the height of the tree. In the worst case, the cost becomes $O(N)$, by simple broadcasting.

Updates. If the height does not need to be balanced, each split only affects the current leaf node and two nodes that are newly created; i.e. only two DHT lookups are needed. Hence, in this case, the total update cost is $O(logN)$. If the height needs to be balanced, the update cost depends on the degree of restructuring needed to maintain the index structure. Even in its simplest case, where updates simply propagate from leaf to root, an update that affects each one of the $logN$ index nodes along the path to the root would need to be communicated to all their corresponding replicas/caches in the leaf nodes. Since the leaf node being updated has the cache information of the entire path to the root which will be effected, the node can create one single message containing all the updates at all levels and forward this message using a broadcast approach, instead of DHT based communication, to all other physical nodes in $O(N)$ messages. Thus, in TRS schemes, there is a trade-off between the efficiencies of the search and of the update.

Workload. In this case, since the peers are symmetric to each other both in terms of directory-lookup as well as re-direction behavior, both workloads are balanced.

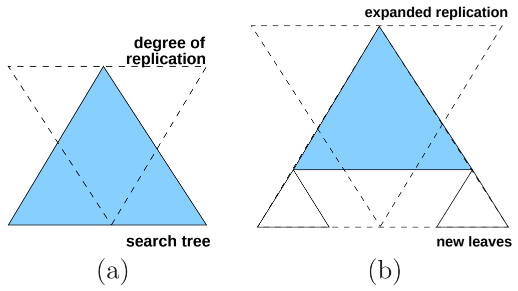

Figure 2. Node replication scheme

6. The Node Replication Scheme

The node replication scheme (NRS) replicates each internal node explicitly, thus ensuring that the load is balanced across the nodes in the network. Thus, unlike the PCS, replication is done at the logical level itself. Figure 2(a) illustrates a conceptual representation of the NRS replication scheme. The filled triangle represents the index tree whereas the dashed triangle represents the corresponding replication graph: for each node the number of replicas is proportional to the number of its leaf descendants; hence the root should have L replicas (where L equals to the number of the leave) while each leaf node has only one copy.

Update. As the tree expands (Figure 2(b)), the replication graph evolves as well. Since the number of times an internal node is replicated depends on the number of the corresponding leaf nodes, the creation of a new leaf node requires replication of all its ancestors. This process, which starts at the leaf and progressively moves towards the root, is depicted in Figure 3. Here, the solid filled circles represent the tree nodes while the dashed circles represent the replicas of the tree nodes explicitly generated. We start with a single root. Each time a node splits, we create one more replica for each one of the nodes along the path from the node to the root. These nodes are randomly assigned to the physical nodes in the system.

If the height does not need to be balanced, each split involves creating one more replica for each node along the path from a leaf to the root, hence the update cost is $O(log^2 N)$. If the height needs to be balanced, this may also require updates to all nodes along the path from the leaf to the root; thus, in addition to the $O(log^2 N)$ replication cost, we also need messages to propagate the updates to the nodes along the path to all their replicas. This can be performed by following parent pointers, using DHT at each level, to communicate the updates to all parents. Since $O(N)$ replicas are involved, and since each DHT communication cost $O(logN)$ messages, the overall update cost is $O(NlogN)$. Alternatively, at $O(logN \times logN)$ time, we can follow one single path from a leaf to the root and prepare one single message containing all updates at all levels of the tree. We can broadcast this combined message to all nodes in $O(N)$ messages. Thus the overall cost of this alternative is $O(log^2 N + N)$.

Lookup. In a height-balanced tree, each search traverses the tree once upward and then downward and each hop along the logical path is equivalent to a DHT lookup cost. Thus the search cost is $O(logN \times logN) = O(log^2 N)$. In a non-height-balanced tree, the worst case search cost is $O(h \times logN)$, where h is the height. These are similar to the lookup costs of PCS solutions. Note that, as in PCS, in general, not all steps of the search has to be performed explicitely. If during the traversal , one reaches a peer which can act as a shortcut towards the subdestination or the final target, this can shorten the lookup process. Since, the number replicas increases with the level of the tree, this means that most searches will be much shorter.

Workload. NRS is designed for explicit balancing of the load in the system. Each node maintains information about only four other nodes: two of its parent's replicas, left child and right child. Whenever a query traverses along the tree path, each node randomly picks one of its two parents for routing in the upward direction. Since, at each replication step, we randomize the selection of the pointers to replicas while ensuring that each parent replica is accessible from the same number of child replicas (and vice versa), the request load flowing over an index node is uniformly distributed among all its logical replicas. Also, given that each internal node is replicated as many times as the corresponding leaves, overall, a uniform load distribution is achieved across all replicas. Since all replicas in the system are randomly mapped to the physical space, this leads to the uniform usage of the physical nodes.

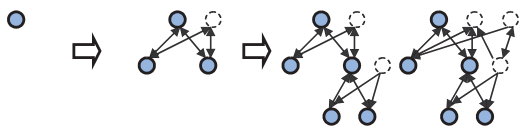
Figure 3. Progressive algorithm to construct the replication graph; each new node contributes to explicit replication at the logical level

7. Evaluation of the Three Schemes

In this section, we evaluate and compare the three mapping approaches under different resource brokerage scenarios. For this purpose, we built a simulation environment using a discrete event-driven simulator called PARSEC developed at UCLA. We used a k-d tree, which does not force height-balancing, as the logical space and Chord [17] as the physical space.

7.1. Setup and Evaluation Metrics

There are three entities: the brokers, the users, and the resource providers. Each broker handles a fixed number of users and providers, in our case, 20 each. Hence there is a 1-to-many mapping between the broker and the users or resource providers. This mapping is pre-defined for simulation purposes. In the experiments, we varied the number of brokers from 50 to 800. Whenever we vary the number of brokers, the numbers of users and providers are also varied implicitly. Thus, the numbers of users and resource providers vary from 1000 to 16,000 accordingly. All the brokers form a Chord ring and the routing between them follows the Chord routing scheme. In the TRS, we placed a replica at each broker, that is, the number of replicas is $O(N)$. We distributed the resources in the system uniformly and focused on how the three approaches perform under different query patterns. Each user query is sent by a user to its dedicated broker. The query is resolved within the broker layer according to the three schemes and the success or failure result is returned to the user.

The first metric for evaluation we used is the length of the search path. A search path is defined as the number of hops (in the Chord network) that it takes for a query to be sent to a broker and returned from the broker, be it a "success" or a "failure". Note that the number of messages exchanged along the path is equivalent to the number of hops that are visited. Other metrics for evaluation includes the update propagation cost and the average workload of the physical nodes.

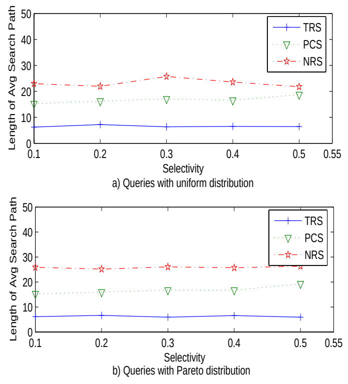

Figure 4. Length of avg. search path for queries with different selectivities and distributions

7.2. Effects of the Query Distribution

The query patterns are determined by two factors: the point distribution and the selectivity. We considered two point distributions: the uniform distribution and a skewed, Pareto, distribution. A skewed query distribution is used to account for query scenarios such as preferences for high end computer systems. The range selectivities, i.e. the query window size, varied from 0.1 to 0.5 with 0.1 increments.

Figure 4 shows the length of average search path using the three schemes when we have queries of different selectivity or distribution. Figure 4(a) shows the length of average search path when queries follow a uniform distribution. Figure 4(b) shows the result when queries follow a Pareto distribution. As shown in these figures, the distribution of the queries does not affect the length of the search path significantly. In a usual range search, selectivity affects the number of data points returned and hence the search path. In our case, however, we are not interested in all but one instance which falls into the range. As long as we can find one available resource that satisfies the user request, the search stops. Hence the selectivity has no significant effect on the length of the average search path.

Since query pattern has little effect on the search path, in the rest of the section, we fix one query pattern (uniform distribution with 0.1 selectivity) to investigate the scalability of the three schemes.

7.3. Scalability

In this section, we investigate three aspects of scalability: look-ups, updates, and workload.

Lookups: Figure 5 shows the length of average search path for the three schemes as the number of brokers increases. TRS is clearly the best. PCS performs better than NRS as internal index nodes are not explicitly

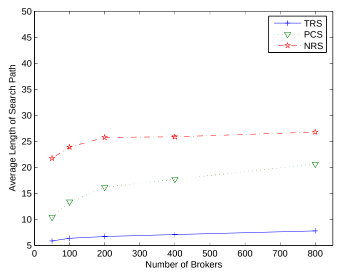

Figure 5. Length of search path for the three schemes as the number of brokers increase

replicated, but cached at the leaves. However, all three schemes show good lookup scalability. As the number of brokers increases, the length of the search path shows a logarithmic growth as expected.

Updates: The update cost consists of two components: (1) the messages exchanged to enable the update and let the relevant brokers be aware; and (2) the physical movement of index data from the original leaf node to the two newly allocated leaf nodes. For all schemes the second component is the same; thus we exclude it from the discussion. Figure 6 shows the average update cost in terms of messages exchanged as the number of brokers increases. TRS shows a linear update cost: since any change to the tree structure has to be propagated to all nodes in the network, it clearly is not scalable. PCS provides the best update performance since change effects are only local. The update cost of NRS is slightly higher as the update requires replication of the nodes along the path to the root. Note that, as updates to the resource availabilities are more or less uniformly distributed in the network, the worst-case behavior of a non-height balanced k-d tree is not observed, except for the TRS scheme. While, the linear update cost of TRS makes it unsuitable for large networks where updates are common, both PCS and NRS show good scalability in terms of update costs. This is especially good considering that, in the resource brokerage framework updates are as likely as queries and the maintenance cost of a height-balanced tree would be prohibitive.

Workload: The third aspect we consider in scalability is workload. As the network size increases, the number of users (and hence the total number of queries generated in the system) increases along with the number of brokers. A good workload scalability would require that as the network (brokers and user queries) grows, the workload on each of the brokers stays stable. Figure 7 shows the average workload on each broker (the number of messages) as the network grows. All three schemes show good scalability. The TRS approach pro-

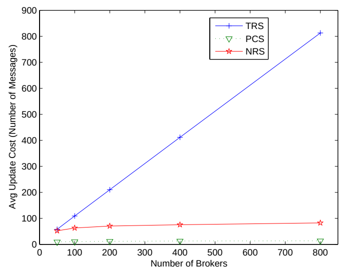

Figure 6. Average update cost: number of messages exchanged as number of brokers increases

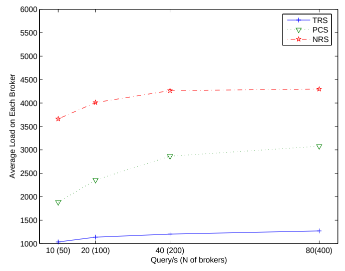

Figure 7. Average workload on each broker as the number of brokers increases

vides best scalability; however, since the update cost of $O(N)$ replicas in this scheme is significantly high, this scheme is not practical. Among the two request- and update-scalable solutions (PCS and NRS), PCS provides a lower average workload on the nodes.

8. Conclusion

We have proposed a general framework that maps tree-structured logical search space for range indexing onto a DHT-based physical node space, with aim of answering point-in-range queries (i.e., resource brokers in a Grid architecture). We have provided three mapping schemes: the tree replication scheme (TRS), the path caching scheme (PCS), and the node replication scheme (NRS). As shown by analysis as well as experiments, these three schemes have different performance characteristics in query cost, update cost, and workload distribution. In general, the TRS scheme may not be applicable to update-heavy environments, like Grid resource brokerage services. PCS and NRS are both scalable and present comparable performance under different resource brokerage scenarios.

References

[1] K. Aberer et al. Advanced peer-to-peer networking: The P-Grid System and its Applications. *PIK Journal, Special Issue on P2P Systems*, 2003.

[2] Artur Andrzejak and Zhichen Xu. Scalable, efficient range queries for grid information services. In *Proceedings of the 2nd IEEE P2P*, pages 33–40, 2002.

[3] James Aspnes and Gauri Shah. Skip graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, 2003.

[4] M. Castro, M. Costo, and A. Rowstron. Debunking some myths about structured and unstructured overlays. In *NSDI*, 2005.

[5] Adina Crainiceanu et al. P-Ring: An Index Structure for Peer-to-Peer Systems. Cornell Univ., Comp. and Info., Scieence TR2004-1946, 2004.

[6] Adina Crainiceanu et al. Querying Peer-to-Peer Networks Using P-Trees. In *WebDB'2004*, 2004.

[7] I. Foster and A. Iamnitchi. On death, taxes, and the convergence of grid and p2p computing. In *Proceedings of the International Workshop on P2P Systems*, 2003.

[8] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The Int. Journal of Supercomputer App. and High Perf. Computing*, 1997.

[9] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems. In *VLDB)*, 2004.

[10] J. Gao and P. Steenkiste. An adaptive protocol for efficient support of range queries in DHT-based systems. In *ICNP'04*, 2004.

[11] A. Gupta, D. Agrawal, and A.El Abbadi. Approximate Range Selection Queries in Peer-to-Peer Systems. In *CIDR03*, pages 141–151, 2003.

[12] N. Harvey et al. Skipnet: A scalable overlay network with practical locality properties. In *USITS03*, 2003.

[13] Adriana Iamnitchi and Ian Foster. On fully decentralized resources discovery in grid environments. In *Workshop on Grid Computing*, 2001.

[14] O.Sahin, A.Gupta, D.Agrawal, and A.El Abbadi. A Peer-to-peer Framework for Caching Range Queries. In *ICDE'2004*, pages 165–176, 2004.

[15] S. Ratnasamy, J. Hellerstein, and S. Shenker. Range queries over DHTs. In *Technical Report IRB-TR-03-009, Intel Corp.*, 2003.

[16] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 161–172. ACM Press, 2001.

[17] Ion Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, 2001.

[18] E. Tanin, A. Harwood, and H. Samet. A distributed quadtree index for peer-to-peer settings. In *ICDE*, 2005.

[19] X. Zhang, J.L. Freschl, and J.M. Schopf. A performance study of monitoring and information services for distributed systems. In *HPDC03*, pages 270–282, 2003.