Range queries in trie-structured overlays*

Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, Karl Aberer
Ecole Polytechnique Fédérale de Lausanne (EPFL)
CH-1015 Lausanne, Switzerland

Abstract

Among the open problems in P2P systems, support for non-trivial search predicates, standardized query languages, distributed query processing, query load balancing, and quality of query results have been identified as some of the most relevant issues. This paper describes how range queries as an important non-trivial search predicate can be supported in a structured overlay network that provides $O(\log n)$ search complexity on top of a trie abstraction. We provide analytical results that show that the proposed approach is efficient, supports arbitrary granularity of ranges, and demonstrate that its algorithmic complexity in terms of messages is independent of the size of the queried ranges and only depends on the size of the result set. In contrast to other systems which provide evaluation results only through simulations, we validate the theoretical analysis of the algorithms with large-scale experiments on the PlanetLab infrastructure using a fully-fledged implementation of our approach.

1. Introduction

Some of the key open research questions for P2P systems [11, 14] include:

- Support for non-trivial search predicates like range queries, operators other than equality, and structured queries.
- Query languages, including definition and expressiveness.
- Distributed query processing.
- Quality of query results: Comprehensiveness, existential quantification, and QoS of results.

*The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322 and was (partly) carried out in the framework of the EPFL Center for Global Computing and supported by the Swiss National Funding Agency OFES as part of the European project Evergrow No 001935.

This paper describes how range queries as an important non-trivial search predicate can be supported in a structured overlay network, implementing DHT functionality with $O(\log n)$ search complexity on top of a physically hierarchy-less overlay abstracted as a logical trie.

Tries are a standard indexing structure from databases to support non-trivial search predicates: *"Tries are a generalization of trees. A trie is a tree for storing strings in which there is one node for every common prefix. The strings are stored in extra leaf nodes."* (Definition by the National Institute of Standards and Technologies). A key benefit of tries is that they cluster semantically close data items which is a critical pre-condition for efficient processing of range queries. If semantically close data items are stored in a highly fragmented manner, as in standard DHT approaches, then their logical proximity is not utilized which significantly impairs the complexity and efficiency of such queries. Traditional DHTs such as Chord or Pastry use uniform hashing functions to map application keys to their identifier space. While this achieves good storage load balancing and efficient discovery of exact keys, it is in conflict with preserving the semantic proximity as it destroys existing relations among the application-specific keys. Keys which are semantically close at the application level are heavily fragmented in a DHT. Thus in such DHTs an additional level of indexing is required to locate semantically close data, like in the simple case of range queries (not to mention more complex predicates). While there are several proposals to do this in existing DHTs, the most generic and usable proposal so far has been to superimpose a trie (prefix hash tree [18]) onto a DHT. However, this approach is inefficient for exactly the reasons given above as semantically close data items are not necessarily stored close to each other in the DHT (high fragmentation), and hence, multiple overlay network queries are required to locate all the content.

In contrast to this family of approaches, we make the overlay network itself the trie, i.e., using "in-network indexing," instead of superimposing a trie onto an existing DHT. Thus we make the overlay network as efficient as possible for range queries and other higher-level predicates while

still offering $O(\log n)$ complexity for exact searches. Essentially, this could be done for any DHT, by using an order-preserving hash function, i.e., $\forall s_1, s_2 : s_1 < s_2 \Rightarrow h(s_1) < h(s_2)$, instead of uniform hashing. However, as the order relation of keys is preserved, the trie's shape adapts to the distribution of the keys and this can create a highly skewed key space partitioning. Most DHT proposals are efficient only when the key space partitions are fairly uniformly distributed, which, however, would be in conflict with storage load-balancing for skewed key distributions, as it is the case when key ordering is preserved. The trie-based P-Grid system [1, 4] (http://www.p-grid.org/) accounts for such problems at the core already.

Range queries over different attributes is a related topic. Multi-dimensional range queries require indexing of more dimensions, e.g., as done in Mercury [9], and we do not explicitly address this issue in this paper. However, we do envisage that different applications may indeed need indices for different (possibly newly derived) attributes, and thus fast construction of a corresponding index may be necessary [5]. So to say, just like Mercury, multiple tries to support range queries over different dimensions may also be supported using multiple P-Grids.

P-Grid uses randomized routing and provides efficient search (logarithmic in terms of key space partitions) irrespective of the tree shape [2, 3]. The trade-off is that more sophisticated emergent mechanisms are required for the fast construction of the trie-structured overlay network in a load-balanced manner [5]. However, the advantages of this approach include the efficiency and the benefits of tries such as simplified support and implementation of non-trivial search predicates. An additional advantage over most other structured overlay networks, which can be viewed as interconnection networks trying to primarily optimize communication costs, is that by taking a more database-oriented perspective, a wealth of well-researched and highly optimized algorithms from databases can be adapted and used to implement data management functionalities on top of P-Grid.

In this paper we show how a trie-structured overlay network can be used to efficiently implement range queries. We start with a brief overview of the principles of the basic system, elaborating properties we rely on for the efficiency of range queries, for example, logarithmic search complexity even for skewed distributions. We then describe the range query algorithms that can be integrated on top of the overlay network and provide analytical results that show that the approach supports arbitrary range granularity and is efficient. We show that the communication cost for executing a range query is independent of the size of the queried range and only depends on the size of the result set. We validate the theoretical results obtained from the analysis of our range-query algorithms with a number of experiments implemented in our Java-based implementation of the basic P-Grid system. The experiments were conducted on the PlanetLab infrastructure showing the practical applicability and efficiency of our approach under real-world networking conditions which incur network churn and high peer load. In contrast to other approaches on range queries, which are primarily validated via simulation, we thus do not have to take any (possibly idealistic or wrong) assumptions on the environment when performing the experiments, but can provide "hard results" from tests based on an implementation that has to account for the combination of all possible problems occurring in a real-world setting.

Using PlanetLab as the experimental test-bed has its caveats, however. We would like to point out that the current experimental evaluation is still limited to the moderate number of available peers, and hence the experiments are limited particularly with respect to assessing the practical scalability of our approach. This is precisely where our theoretical analysis in Section 3 comes into play as based on the results of the experiments and the analytical model, we can at least justify that our approach is likely to scale which we can immediately verify as soon as larger-scale deployments can be done. Additionally, the experimental environment on PlanetLab cannot be controlled. However, PlanetLab is *the real world* and newly developed algorithms do not only have to look good on paper, backed up by analysis and simulations, but at the end of the day they have to prove their efficiency and applicability in a real-world scenario. Given this, we can conclude that our algorithms actually seem to be be quite robust even in a real environment with churn caused by the unpredictable behavior of PlanetLab nodes. Thus a theoretical analysis, which is absolutely necessary for a first evaluation of any approach, and a practical experimental validation as presented in this paper, complement each other to really demonstrate the applicability of approaches that deal with large-scale distributed systems, which is typically not possible in an approach solely based on theory and simulations, since both theoretical and simulation models are idealized, and can never reflect all the properties of a true large-scale distributed system deployed over a network.

2. P-Grid: A trie-structured overlay

We use the P-Grid [1, 4] DHT to evaluate the approach presented in this paper. We assume that the reader is relatively familiar with the standard DHT approach and thus we only present P-Grid's distinguishing properties. As any DHT approach, P-Grid associates peers with data keys from a key space, i.e., partitions of the underlying distributed data structure. Each peer is responsible for some part of the overall key space and maintains routing information to forward queries and requests.

Without constraining general applicability we use binary

keys in the following. This is not a fundamental limitation as a generalization of the P-Grid system to k-ary structures is natural, and exists [6]. P-Grid peers refer to a common underlying binary trie structure in order to organize their routing tables (as opposed to other topologies, such as rings [24], multi-dimensional spaces [19], or hypercubes [23]). In the following we will use the terms trie and tree conterminously.

In P-Grid each peer $p \in P$ is associated with a leaf of the binary tree. Each leaf corresponds to a binary string $\pi \in \Pi$, also called the *key-space partition*. Thus each peer p is associated with a path $\pi(p)$. For search, the peer stores for each prefix $\pi(p,l)$ of $\pi(p)$ of length l a set of references $\rho(p,l)$ to peers q with property $\overline{\pi(p,l)} = \pi(q,l)$, where $\overline{\pi}$ is the binary string π with the last bit inverted. This means that at each level of the tree the peer has references to some other peers that do not pertain to the peer's subtree at that level which enables the implementation of prefix routing for efficient search. The cost for storing the references and the associated maintenance cost scale as they are bounded by the depth of the underlying binary tree.

Each peer stores a set of data items $\delta(p)$. For $d \in \delta(p)$ the binary key $key(d)$ is calculated using an order-preserving hash function. $key(d)$ has $\pi(p)$ as prefix but it is not excluded that temporarily also other data items are stored at a peer, that is, the set $\delta(p, \pi(p))$ of data items whose key matches $\pi(p)$ can be a proper subset of $\delta(p)$. Moreover, for fault-tolerance, query load-balancing and hot-spot handling, multiple peers are associated with the same key-space partition (structural replication), and peers additionally also maintain references $\sigma(p)$ to peers with the same path, i.e., their replicas, and use epidemic algorithms to maintain replica consistency. Figure 1 shows a simple example of a P-Grid tree.

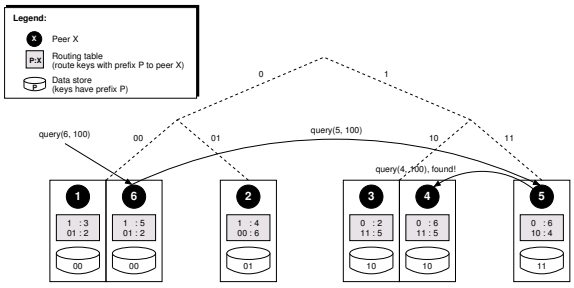


Figure 1. P-Grid overlay network

P-Grid's hash function maps application keys to binary strings. In the reference implementation we assume application keys to be strings for simplicity, but in fact any data type can be used. The hash function is order-reserving, i.e., it satisfies the following property for two input strings s_1 and s_2:

$$s_1 \subseteq s_2 \;\Rightarrow\; key(s_1) \subseteq key(s_2)$$

where $\subseteq$ means *is-prefix-of*.

To enable this mapping, we first constructed a balanced trie from a sample string database consisting of unique, lexicographically sorted strings of equal length (sample string databases can be provided by the user). The database is recursively bisected into equally-sized partitions until each partition is smaller than a threshold. The keys P-Grid uses are then calculated by using the application key to "navigate" character-wise through this trie and appending "0" to the generated key for each "left-turn" or "1" otherwise.

P-Grid, like any other DHT approach, supports two basic operations: *Retrieve(key)* for searching a certain key and retrieving the associated data item and *Insert(key, value)* for storing new data items. Since P-Grid uses a binary tree, *Retrieve(key)* is of complexity $O(\log |\Pi|)$, measured in messages required for resolving a search request, in a balanced tree, i.e., all paths associated with peers are of equal length. Skewed data distributions may imbalance the tree, so that it may seem that search cost may become non-logarithmic in the number of messages. However, in [2, 3] it is shown that due to the randomized choice of routing references from the complimentary sub-tree, the expected search cost remains logarithmic ($0.5 \log n$), independently of how the P-Grid is structured. The intuition why this works is that in search operations keys are not resolved bit-wise but in larger blocks thus the search costs remain logarithmic in terms of messages. This is important as P-Grid uses order-preserving hashing to compute keys which may lead to non-uniform key distributions.

The basic search algorithm is shown in Algorithm 1. p in the algorithm denotes the peer that currently processes the request.

Algorithm 1 Search in P-Grid: Retrieve(key, p)

```
1: if π(p) ⊆ key then
2:     return(d ∈ δ(p)|key(d) = key);
3: else
4:     determine l such that π(key, l) = π(p, l)‾;
5:     r = randomly selected element from ρ(p, l);
6:     Retrieve(key, r);
7: end if
```

The algorithm always terminates successfully, if the P-Grid is complete (ensured by the construction algorithm) and at least one peer in each partition is reachable (ensured through redundant routing table entries and replication). Due to the definition of ρ and $Retrieve(key, p)$ it will always find the location of a peer at which the search can continue (use of completeness). With each invocation of $Retrieve(key, p)$ the length of the common prefix of $\pi(p)$ and key increases at least by one and therefore the algorithm always terminates. Note that, while the network has a tree/trie abstraction, the system is hierarchy-less, and all peers reside at the leaf nodes.

Insert(key, value) is based on P-Grid's more general update functionality [12] which provides probabilistic guarantees for consistency and is efficient even in highly unreliable, replicated environments, i.e., $O(\log |\Pi| + replication\ factor)$. An insert operation is executed in two logical phases: First an arbitrary peer responsible for the key-space to which the key belongs is located (*Retrieve(key)*) and then the found peer notifies its replicas about the inserted *key* using a light-weight hybrid push-and-pull gossiping mechanism.

There are several other DHTs which topologically resemble P-Grid and use prefix-based routing variants, for example, Pastry [21] and particularly Kademlia [16] whose XOR distance metric results in the same tree abstraction and choice of routes from all peers in complementary subtrees as in P-Grid. The important distinguishing features of P-Grid include the emergent nature of the P-Grid network based on randomized algorithms, support for substring queries, the disentanglement of peer identifiers from the associated key space, and the adaptive, structural replication (multi-faceted load-balancing of storage and query load) [5].

There is another motivation for having a trie-structured overlay network instead of a standard distributed hash table: The real advantage of traditionally using a hash table in main memory is the constant time of lookup, insert, and delete operations. But to facilitate this, a hash table sacrifices the order-relationship of the keys. However, over a network, where only parts of the hash table are stored at each location, we need multiple overlay hops anyway. For most conventional DHTs the number of hops is logarithmic in the network size. Thus the main advantage of constant-time access no longer exists in DHTs. This made P-Grid a natural choice for us to use it as the underlying routing network to support range queries, since it provides normal key search for same order of message complexity as a DHT, but in addition can be naturally extended to support range queries.

The range query approach which is presented in the following is integrated in our P-Grid implementation which is available at http://www.p-grid.org/.

3. Range query algorithms and complexity analysis

As described in the previous sections, P-Grid uses an order-preserving hash function. Thus the resulting P-Grid tree in fact is a trie. This may lead to skewed data distributions despite which P-Grid can still guarantee logarithmic search complexity. Order-preserving hash functions, however, enable prefix queries and thus range queries of arbitrary granularity can be processed efficiently as well in P-Grid., i.e., $O(\log |\Pi| + |\{p \mid \kappa \subseteq \pi(p)\}|)$, where κ denotes the common prefix of the borders of the queried range. We will discuss two classes of algorithms: the *min-max traversal* algorithm which is sequential and the *shower* algorithm which parallelizes the execution of range queries.

Min-max traversal algorithm: Range queries can be processed sequentially by starting from a peer holding data items belonging to one bound of the range and forwarding the query to a peer responsible for the next partition of the key space, until a peer responsible for the other bound of the range is encountered. This strategy is called *min-max traversal*. The underlying data structure itself does not always have the information about peers belonging to the next neighboring key space partitions. However, such routes can be established either during the construction of the P-Grid overlay structure (algorithmically trivial), or at run-time using the existing routing information at the peers. Figure 2(a) shows the min-max traversal algorithm graphically.

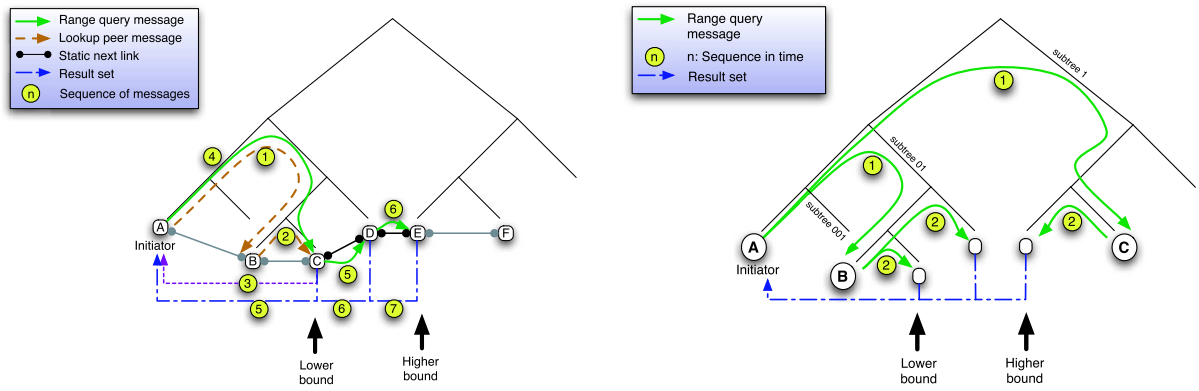
First peer A initiates the range query by querying P-Grid for the lower bound of the range which is peer C in this example. Steps (1) and (2) denote standard P-Grid routing and in step (3) the result is returned to peer A, i.e., peer C. Then in step (4) peer A sends the range query request to peer C and peer C sends its data pertaining to the interval to peer A (in the implementation steps (3), (4), and (5) are actually done in one step). Concurrently the range query is forwarded to peer D using the "next" pointer. Peer D checks whether it is in the queried range, and if yes, peer D sends its data pertaining to the interval to peer A, and concurrently forwards the range query to peer E which repeats the same operations as peer D except that it does not forward the query to another peer as it has checked that it is a peer responsible for the other bound of the queried range.

Algorithm 2 shows this algorithm in pseudo code.

Algorithm 2 Sequential range queries: minmax(R, p)

```
1: if π(p) ⊆ R then
2:     return(d ∈ δ(p)|key(d) ∈ R);
3:     determine a peer r such that r is responsible for the next key space
       partition;
4:     minmax(R, r);
5: end if
```

For simplifying the analysis we assume that the algorithm starts at the lower bound of the range R (the routing of the query to the lower bound is not shown here, but is algorithmically trivial in P-Grid). It is assumed that the neighbor links are cached at each peer during the construction of the trie (this is also algorithmically trivial). In the complexity analysis of this algorithm we can assume storage load-balancing (which is achieved stochastically by the P-Grid base system) and that on average there exist M data items per key space partition. Then, if there is a range query for the range R, such that there are D data items in the given range, search cost and latency using min-max traversal (as-



(a) Min-max traversal

(b) Shower

Figure 2. Range query strategies

suming "next" links have been established during construction) is $O(\log_2 |\Pi|) + |\Pi_R| - 1$, where $|\Pi_R|$ is the number of partitions over which the whole range is stored in P-Grid and $|\Pi|$ is the total number of leaf-nodes in the complete P-Grid tree (total number of key space partitions). The search cost and latency using min-max traversal is dependent on the size of the answer set D for the range query, but independent of the size of the range R of the query. This is because $|\Pi_R|$ has an expected value of D/M, and in particular, using Markov's inequality, $Pr[|\Pi_R| \geq cD/M] \leq \frac{1}{c}$ for any positive c thus giving a weak bound on the deviation. We do not consider the trivial case $D \leq M$ as this would only affect 1 or 2 peers and concentrate on the more general case of $D > M$.

As already mentioned, establishing and maintaining "next" pointers in P-Grid is algorithmically trivial and most other DHTs proactively maintain it as well. Without them, an additional small overhead of $|\Pi_R| O(\log_2 |\Pi|)$ would have to be included. Note that this is an upper bound, as part of the routing does not have to be repeated for the peers in the interval.

Shower algorithm: The other variant for processing range queries is to do them concurrently. Here, the range query is first forwarded to an arbitrary peer responsible for any of the key space partitions within the range, and then the query is forwarded to the other partitions in the interval using this peer's routing table. The process is recursive, and since the query is split in multiple queries which appear to trickle down to all the key-space partitions in the range, we call it the *shower algorithm*. The intuition of the algorithm is shown graphically in Figure 2(b).

In the course of forwarding, it is possible that the query is forwarded to a peer responsible for keys outside the range. However, it is guaranteed that this peer will forward the range query back to a key-space partition within the range. Moreover, the P-Grid routing ensures that no key space partition will get duplicates of the range queries.

Algorithm 3 gives the pseudo code for the *shower* algorithm.

Algorithm 3 Parallel range queries: shower($R, l_{current}, p$)

```
1: if π(p) ⊆ R then
2:    return(d ∈ δ(p)|key(d) ∈ R);
3: end if
4: determine l_l such that π(min(R), l_l) = π(p, l_l);
5: determine l_h such that π(max(R), l_h) = π(p, l_h);
6: l_min = max(l_current, min(l_l, l_h));
7: l_max = max(l_l, l_h);
8: if l_current < l_max then
9:    for l = l_min to l_max do
10:      r = randomly selected element from ρ(p, l);
11:      shower(R, l+1, r);
12:   end for
13: end if
```

The search cost (in terms of messages) of this variant is lower bounded by $O(x) + |\Pi_R| - 1$. Since every message created in the range sub-space reaches a different leaf node (since the sub-spaces are exclusive), and there are expected D/M such sub-spaces, the upper bound is $O(x) + min(2O(|\Pi_R|), 2^{Depth-x})$ where $Depth$ is the maximum path length of any partition in the range. Thus the complexity of the shower algorithm is again dependent only on the size of the answer set D for the range query, but independent of the size of the range R of the query.

The upper bound for latency is $O(x)+O(Depth-x)$. In particular, unlike in the sequential variant, the latency of the parallelized shower algorithm is independent of the number of data items in the range R, but depends on the distribution of the data items (which determines the $Depth$). Note that the issuer of the query will start getting responses for part of the range with a minimum latency of $O(x)$, since it will already encounter some peer responsible for part of the

range.

The expected value of x is $0.5 \log (nM/D)$. The intuition for the value of x is that, if we increase the average memory of each logical partition to D instead of M, there will be $\frac{n}{D/M}$ key space partitions in total, otherwise retaining the routing network's properties, and since first the query needs to reach any arbitrary peer within the range, this translates into reaching this virtual partition of average size D, and hence x is the expected search cost in this new network, which has the same topological properties, but fewer (nM/D) partitions.

4. Experiments

The two range query algorithms were implemented on top of the Java-based P-Grid implementation (available from http://www.p-grid.org/) and we performed a number of large-scale experiments on PlanetLab [10] to validate the analytical results presented in Section 3 in a practical setting.

4.1. Experimental setup

In the experiments we used a network of 250 peers each running on a dedicated physical PlanetLab node. We inserted 2500 unique data items into the system and required an average replication factor of 5 which is necessary in any overlay network to compensate for node and communication failures. Thus initially we would have a total of $5 * 2500 = 12500$ data items in the system and each peer would be responsible for $5 \frac{2500}{250} = 50$ data items. The real number of the data items in the system in fact was higher as for load-balancing each peer was required to manage a minimum of 50 and a maximum of 100 data items, and given the randomized construction approach of P-Grid, each peer would thus hold on average 75 data items, i.e., the total number of data items in the system was $250 * 75 = 18750$.

To show that the algorithms basically work for any data distribution, we used two different data sets, one uniformly distributed and one Pareto distributed (with a probability density function of $\frac{a\ k^a}{x^{1+a}}$ and parameters $k = 1$ and $a = 2.0$) as shown in Figure 3.

Pareto is a typical long-tail distribution which occurs frequently. We will see in the experiments that P-Grid is insensible to such distributions due to the efficiency of the underlying load-balancing algorithm which balances both storage and replication load. We can thus safely infer that if the results are good for a Pareto distribution, the system will perform equally well for other frequent long-tail distributions, e.g., Zipf.

In the experiments each peer selected randomly 10 data items of a data global set according to one of these distributions. The peers then constructed a P-Grid which had an average height of $\log_2 \frac{2500}{10*5} = 5.6$. Then range queries which affected data from all partitions of the data sets were issued. The queries were started from randomly chosen peers with random lower range bounds, and were constructed in a way, such that they would return 50, 100, 150, 200, 400, and 800 data items. For each of the six answer set sizes, each of the two distributions, and each of the two algorithms, one query was issued by each of the 250 peers, i.e., a total of $6 * 2 * 2 * 250 = 6000$ queries resulting in 250 values per data point in Figures 4–7.

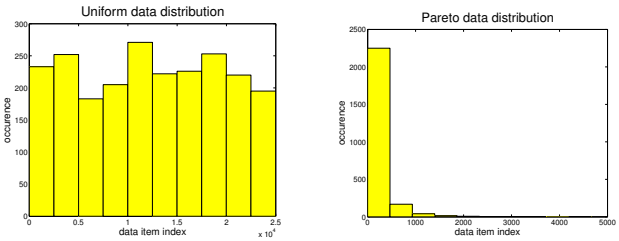


Figure 3. Data set distributions

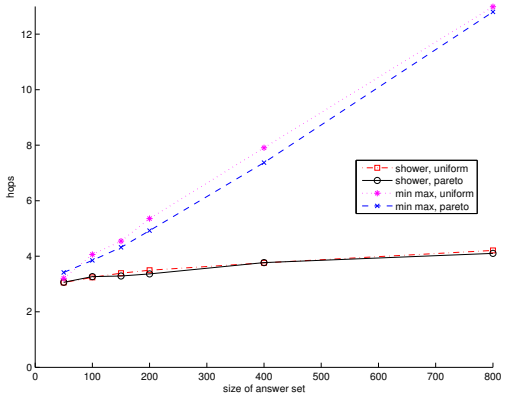
4.2. Experimental results

There are several performance metrics of interest to evaluate the system as well as the algorithms for their suitability to support range queries. This includes load-balance characteristics (storage, replication, and query load), data fragmentation, as well as message costs and latency for various data distributions. Previous studies [5] have shown P-Grid's efficient multi-faceted load-balancing characteristics and that the use of order-preserving hashing ensures low data fragmentation, while the dynamic construction of the trie structures ensures storage-load balancing. We could also verify this in our experiments.

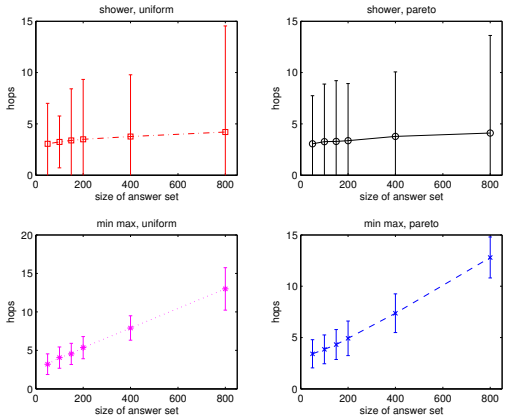
Thus the main objectives of our experiments in this paper were to demonstrate the cost/latency trade-off of the range query algorithms, and to show that because of the use of a load-balanced trie-structured overlay network, the cost of range queries is independent of the data distribution and the size of the range, but only dependent on the used algorithm and the size of the answer set which we expected from the theoretical analysis of Section 3. From the experimental results presented in the following, we can observe that the cost and latencies are indeed independent of the distribution and indirectly prove that the overlay network has good storage-load balancing characteristics.

Figure 4 shows the costs incurred by range queries in terms of message latency (hops), i.e., the maximum number of messages required to hit each sub-partition of the range, i.e., one peer in each sub-partition. Figure 4(a) shows a direct comparison of the experimental results and Figure 4(b)

gives the standard deviations of each of the four types of experiments as error bars.

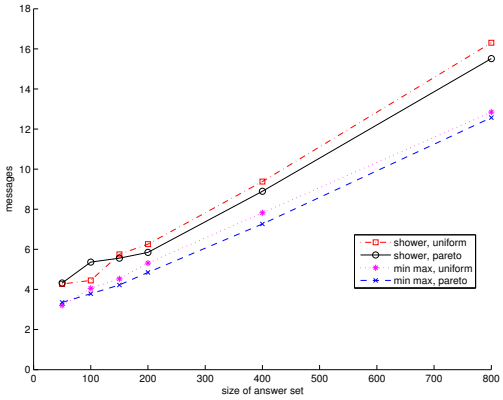


(a) Comparison

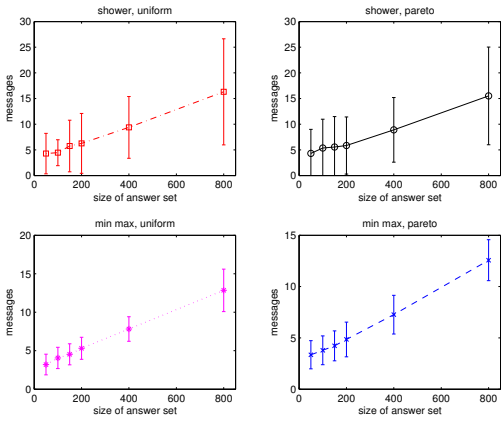


(b) Standard deviation

Figure 4. Message latency (hops)



(a) Comparison



(b) Standard deviation

Figure 5. Message cost

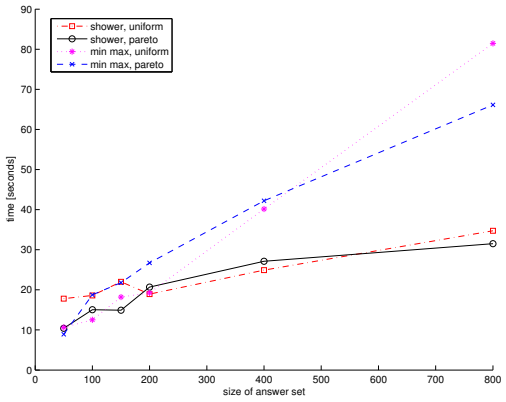
On average we need 3 hops to reach a responsible peer for both types of algorithms, but the min-max algorithm then suffers from the sequential traversal of the range to reach all sub-partitions after reaching the lower bound. This leads to increasing hop counts with increasing range sizes whereas for the shower algorithm the number of hops remains constant, i.e., it is rather insensitive to the size of the answer set as an increase in the number of hops for this algorithm basically means that the range has exceeded one level in the tree and an additional hop is necessary as the "shower" has to start at the next higher level. However, this benefit comes at the cost of an increase in the overall messages as shown in Figure 5. Figure 5(a) shows a direct comparison of the experimental results and Figure 5(b) gives the standard deviations of each of the four types of experiments as error bars.

The shower algorithm requires a slightly higher number of messages but improves latency as it sends them to the responsible peers in parallel. Therefore all peers responsible for a range section are reached after 3 hops (in the experiment's setup) independent of the range size. Range queries with an answer set size of 50 are answered mostly by one peer because peers on average are responsible for 75 data items. It can further be seen that both algorithms perform equally well for both data distributions and scale well as expected. An increase of the answer set size by a multiplicative factor of the average peer storage size yields an additional message on average which is the best possible result achievable with limited storage available at the peers and again indirectly proves the optimal behavior of the underlying load-balancing algorithm.

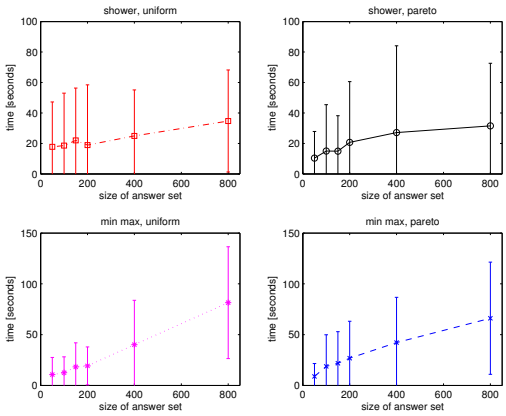
Figure 5 also shows the total number of peers involved in a range query, i.e., the number of peers forwarding or replying to a range query. For the min-max algorithm this number is equal to the number of messages because only one message is first routed to the lower bound and then forwarded to the higher bound. Therefore the number of peers forwarding a query to a peer of the desired range is smaller

than for the shower algorithm. More peers are involved during the shower algorithm because messages are sent in parallel to reach desired peers (partitions).

In terms of query latency, it is interesting to see that the shower algorithm is almost insensible towards answer set sizes. As can be seen in Figure 6 the latency is nearly constant.



(a) Comparison



(b) Standard deviation

Figure 6. Query latency (time)

This can be explained by the fact that a considerable number of data items would have to be added before the trie increases its height which is the major contribution to the latency for this algorithm. For the min-max case the latency increases for obvious reasons as messages are forwarded sequentially which increases the latency. Here an increase of the height of the trie has a much more dramatic influence as the min-max algorithm heavily depends on the width of the interval. While increasing the height of the trie means only an additional hop for the shower-algorithm which is processed largely in parallel, for the min-max algorithm the number of sequential messages increases by a factor of 2 on average. Note that this is expected from theory, since the height of the tree will increase by 1 only if approximately twice the data items are in the same range, and in the min-max algorithm, both latency and message cost is proportional to the number of data-items in the answer-set.

A side result which can be inferred from these plots is that the smallest range queries involving 3–5 peers take approximately 10–20 seconds on average. Larger range queries using the min-max algorithm take a multiple of that. This can be explained by the success of PlanetLab as an experimental test-bed, since a large number of experiments are conducted concurrently which considerably slows down PlanetLab's overall performance.

Finally, in Figure 7 we show what level of result completeness we could achieve by our range queries.

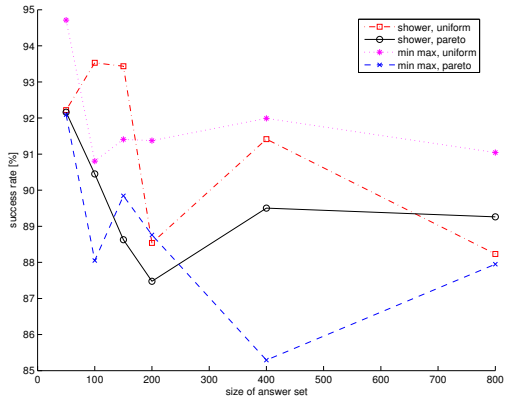


Figure 7. Result completeness

This measure represents the percentage of received data items as answers to a range query with respect to the actual number of data items inserted (present) in the specific range. The result completeness is around 90% and is mainly independent of the range sizes and the data distributions. We observed several problems during our experiments in respect to the PlanetLab environment, for example, communication problems and crashes of PlanetLab nodes (not of the tested P-Grid system but the physical PlanetLab nodes), which explain the non-exhaustive results. Note that, while it is an issue that is beyond the scope of this paper (such failures because of unreliable peers are characteristic of any deployed P2P system, the relatively high success rate in fact demonstrates the robustness of P-Grid under churn. Smaller scale experiments in a local environment with lower numbers of nodes and node failures have proven the functional correctness of our implementation and provided a 100% success rate. To increase the success rate on PlanetLab we could increase the replication factor, i.e., data is replicated more often, and thus node failures could be possibly compensated better. This will increase the maintenance overhead but should provide better results. However, due to the duration

of the experiments and the lack of possibility to assess the conditions on PlanetLab that caused a certain experimental result and behavior, we have no experimental evaluation of this strategy yet. In the experiments discussed above we used a replication factor of 5 on average (in fact, each data item was replicated between 1 and 10 times). Taking this into account and the very dynamic situation on PlanetLab a success rate of 90% seems reasonable. In future work, we will explore the possibility to adapt replication to the dynamic situation on the physical network to improve on the result completeness.

5. Related Work

Traditional database research has shown that tries are among the most practical data structures to support range queries. The work on prefix hash trees (PHT) superimposes a P-Grid-like trie onto an arbitrary structured overlay network [18]. The advantage of PHT is thus its universal usability on top of any DHT, however, it is considerably less efficient as we had argued earlier. Using a native trie structure as is done in P-Grid makes range queries more efficient in terms of both message cost and latency. Note that the analysis we did in this paper gives the costs in terms of the total number of overlay network messages. The analysis of PHT provides the number of DHT searches for answering a range query, and each of these DHT searches for a typical DHT (like Chord [24]) involves logarithmic number of messages in terms of the key space partitions (alternatively peer population). This is due to the fact that semantically close data items are not necessarily stored close to each other in the overlay network (high fragmentation), and hence, multiple overlay network queries are required to locate all the content. In contrast, tries cluster semantically close data items which in turn enable efficient range access. Another recent approach [15] uses a hierarchical tree structure but because of the hierarchy, it inherently has poor fault-tolerance and poor query load-balancing characteristics.

To support approximate range queries, locality-preserving hashing to hash ranges instead of keywords is used in [13]. An improvement of this approach to support exact range queries is proposed in [22]. The fundamental problem of these approaches is that the ranges themselves are hashed, and hence, simple key search operations are not supported or are highly inefficient. Since both key and range queries are needed, it is desirable to have one mechanism supporting both, instead of maintaining separate hash table for keys, and separate hash tables for ranges, because such a strategy fails to reuse the resources of the peers. These approaches [13, 22] lead to very bad fragmentation even for related ranges, and can result in either poor storage-load balancing or inefficient access. Moreover, since they use CAN as the underlying network, the search efficiency guarantees hold only for uniform partitioning of the space, which conflicts with storage load which is arbitrarily distributed, as will be the case for caching range queries, more so because queries will also be non-uniformly distributed.

In terms of key search efficiency, support for range queries and storage load-balancing, there are some interesting novel structured overlay network abstractions which exhibit performance comparable to our trie-structured proposal: Skip Graphs [7, 8] which are based on skip lists [17], and Mercury [9] which is based on small-world routing. Skip Graphs can be viewed as a trie of skip lists that share their lower levels. As Skip Graphs preserve the ordering relation among keys they also support range queries. Similar to the shower variant of our algorithms, range queries are resolved by finding any node in the interval ($O(\log n)$ messages) and then broadcasting the query through the m nodes in the interval which requires $O(m \log n)$ messages. In total this is still of logarithmic complexity but quite a bit higher than the effort (in terms of messages) incurred by our approach. Mercury, on the other hand, retains the data sequentially, dynamically assigns the range for which individual peers are responsible in order to provide good load-balancing, and uses small-world routing among the peers. Multiple-attribute range queries by using an individual index for each attribute as proposed in Mercury can be done based on any indexing scheme, including ours. The important and unaddressed issue in all existing literature on multiple-attribute range queries is the issue of efficient joins. Though Skip Graphs and Mercury offer comparable complexity characteristics in terms of search and range queries as our approach, these systems have so far only been evaluated with simulations, and no real implementations or experimental evaluations in a real-world networking scenario exist. For our approach, however, we do not only provide the theoretical study of the performance, but also report on deployment and experimentation of a fully implemented overlay network.

There exist many other range query proposals, which are of lesser relevance than the approaches discussed above. A detailed survey of search mechanisms in P2P systems, including range queries can be found in [20].

6. Conclusions

Non-trivial search predicates such as range queries are an important area of ongoing research to improve the versatility of structured P2P systems. In this paper we have described how range queries can be implemented on top of a trie-structured DHT. A theoretical analysis showed the efficiency of the approach as its algorithmic complexity in terms of messages is independent of the size of the queried

ranges and only depends on the size of the result set while supporting arbitrary range granularity. We evaluated our algorithms for range queries in a practical setting with experiments on the PlanetLab infrastructure. Over the last years of research on P2P systems there have been numerous proposals on paper, but what is increasingly necessary is to have fully-fledged implementations of these systems that can be tested in practice, i.e., pass the only relevant test, which is practical applicability.

References

[1] K. Aberer. P-Grid: A self-organizing access structure for P2P information systems. In *Sixth International Conference on Cooperative Information Systems (CoopIS)*, 2001.

[2] K. Aberer. Efficient Search in Unbalanced, Randomized Peer-To-Peer Search Trees. Technical Report IC/2002/79, Ecole Polytechnique Fédérale de Lausanne (EPFL), 2002.

[3] K. Aberer. Scalable Data Access in P2P Systems Using Unbalanced Search Trees. In *4th Workshop on Distributed Data and Structures (WDAS'2002)*, 2002.

[4] K. Aberer, P. Cudré-Mauroux, A. Datta, Z. Despotovic, M. Hauswirth, M. Punceva, and R. Schmidt. P-Grid: A Self-organizing Structured P2P System. *ACM SIGMOD Record*, 32(3), 2003.

[5] K. Aberer, A. Datta, M. Hauswirth, and R. Schmidt. Indexing data-oriented overlay networks. *31st International Conference on Very Large Databases (VLDB)*, 2005.

[6] K. Aberer and M. Punceva. Efficient Search in Structured Peer-to-Peer Systems: Binary v.s. k-ary Unbalanced Tree Structures. In *International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*, 2003.

[7] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load balancing and locality in range-queriable data structures. In *ACM PODC*, 2004.

[8] J. Aspnes and G. Shah. Skip graphs. In *ACM-SIAM Symposium on Discrete Algorithms*, Jan. 2003.

[9] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting scalable multi-attribute range queries. *SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[10] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *ACM SIGCOMM Computer Communication Review*, 33(3), July 2003.

[11] N. Daswani, H. Garcia-Molina, and B. Yang. Open Problems in Data-Sharing Peer-to-Peer Systems. In *9th International Conference on Database Theory (ICDT)*, 2003.

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

[13] A. Gupta, D. Agrawal, and A. E. Abbadi. Approximate Range Selection Queries in Peer-to-Peer Systems. In *1st Biennial Conference on Innovative Data Systems Research (CIDR 2003)*, 2003.

[14] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks. In *IPTPS*, 2002.

[15] C. Y. Liau, W. S. Ng, R. Panicker, S. Bressan, and K.-L. Tan. Efficient range queries and fast lookup services for scalable P2P networks. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (DBISP2P)*, 2004.

[16] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric. In *IPTPS*, 2002.

[17] W. Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6), 1990.

[18] S. Ramabhadran, S. Ratnasamy, J. M. Hellerstein, and S. Shenker. Brief Announcement: Prefix Hash Tree. In *ACM PODC*, 2004.

[19] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *ACM SIGCOMM*, 2001.

[20] J. Risson and T. Moors. Survey of Research towards Robust Peer-to-Peer Networks: Search Methods. Technical Report UNSW-EE-P2P-1-1, University of New South Wales, Sydney, Australia, Sep. 2004. http://www.ee.unsw.edu.au/~timm/pubs/robust_p2p/submitted.pdf.

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

[22] O. D. Sahin, A. Gupta, D. Agrawal, and A. E. Abbadi. A Peer-to-peer Framework for Caching Range Queries. In *20th International Conference on Data Engineering (ICDE)*, 2004.

[23] M. Schlosser, M. Sintek, S. Decker, and W. Nejdl. Hypercup–hypercubes, ontologies, and efficient search on peer-to-peer networks. *LNCS*, 2530, 2003.

[24] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *ACM SIGCOMM*, 2001.