

# PRoBe: Multi-dimensional Range Queries in P2P Networks\*

O.D. Sahin, S. Antony, D. Agrawal, and A. El Abbadi

Department of Computer Science,  
University of California at Santa Barbara,  
Santa Barbara, CA 93106, USA  
{odsahin, shyam, agrawal, amr}@cs.ucsb.edu

**Abstract.** Structured P2P systems are effective for exact key searches in a distributed environment as they offer scalability, self-organization, and dynamicity. These valuable properties also make them a candidate for more complex queries, such as range queries. In this paper, we describe PRoBe, a system that supports range queries over multiple attributes in P2P networks. PRoBe uses a multi-dimensional logical space for this purpose and maps data items onto this space based on their attribute values. The logical space is divided into hyper-rectangles, each maintained by a peer in the system. The range queries correspond to hyper-rectangles which are answered by forwarding the query to the peers responsible for overlapping regions of the logical space. We also propose load balancing techniques and show how cached query answers can be utilized for the efficient evaluation of similar range queries. The performance of PRoBe and the effects of various parameters are analyzed through a simulation study.

## 1 Introduction

Peer-to-peer (P2P) systems are a popular paradigm for exchanging data among users in a decentralized manner. Currently P2P systems efficiently support exact key lookups in addition to offering properties such as scalability, decentralization, self-organization, and dynamic node insertion and departure. These valuable properties also render them a candidate for more complex queries, such as range queries. Several techniques have been proposed for supporting range queries over a single attribute. Our goal here is to develop a scheme for supporting range queries over multiple attributes. Some example applications that can benefit from a P2P architecture that supports multi-dimensional range queries are as follows: Resource discovery in Grid Computing [1, 2], publish/subscribe systems [3], multiplayer games [4], multi-dimensional data sharing in P2P networks [5], and P2P databases [6, 7].

In this paper, we describe PRoBe (P2P Range Queries over Multiple Attributes using Hyper-Boxes), a P2P architecture that supports range queries

---

\* This research was supported in parts by NSF grants CNF 04-23336, IIS 02-23022, and IIS 02-20152.

over multiple attributes. PRoBe maps data items onto a multi-dimensional logical space. This space is divided into non-overlapping hyper-rectangular zones and each zone is maintained by a peer in the system. Each range query also corresponds to a hyper-rectangle in the logical space and is answered by contacting all the peers whose zones intersect with the query.

Supporting range queries in a P2P system introduces several challenges. Most of the existing systems that are designed for exact key lookups rely on hashing data items. Since hashing destroys locality, we cannot hash data items. In order to support range queries we distribute data based on their values. This distribution, however, may lead to skewed distribution of data in the system because most real world data sets tend to be skewed. Hence we need to employ load-balancing techniques to prevent peers from overloading. We also discuss the utilization of cached query answers for the efficient computation of future range queries. We propose a mapping that enables peers to advertise their cached results and to locate the cached results that are similar to a given query.

The rest of the paper is organized as follows. Section 2 surveys the related work. The design of PRoBe is explained in Sect.3. We describe techniques for load balancing and incorporating caching into the system in Sect.4. Experimental results evaluating different aspects of PRoBe are presented in Sect.5. Section 6 concludes the paper.

## 2 Related Work

Structured P2P systems impose a certain structure on the overlay network and control the placement of data. For example, Distributed Hash Tables (DHTs) [8, 9, 10, 11] hash both peers and objects onto a logical space and assign each object to a peer dynamically. They offer very efficient exact key lookups, which is logarithmic or sublinear in the number of peers. However, hashing prevents DHTs from supporting range queries. Several studies address this problem and support range queries over a single attribute by distributing the data tuples based on their attribute values and using explicit techniques to balance the load over the peers [12, 13, 14]. These load balancing techniques are based on redistributing data items among peer pairs and changing the locations of peers in the logical space, and they achieve constant degree of imbalance ratio among the peers. Another structured P2P system, Skip Graphs [15, 16], supports range queries over a single attribute by organizing the peers into a distributed skip list structure based on their key values. To support range queries, PePeR [17] assigns domain intervals to peers, whereas the inverse Hilbert curve is used for mapping the attribute domain to a  $d$ -dimensional CAN space in [1]. Data structures for prefix search in P2P networks are described in [18, 19].

Recently interest has increased to support multi-dimensional range queries. Ganesan et al. [20] investigate two approaches: 1) *SCRAP* maps both data items and queries from higher dimensions to 1 dimension with space filling curves and then uses existing schemes for routing [15, 16] and load balancing [12], 2) *MURK* uses a multi-dimensional logical space to distribute data items among

peers and to identify peers with data relevant to queries. Mercury [4] creates a separate overlay for each attribute. Range queries are forwarded to a single hub based on node-count histograms constructed through a sampling mechanism. MAAN [2] and Squid [21] map multi-dimensional objects to the Chord identifier space using uniform locality preserving hashing and Hilbert curve, respectively. SkipIndex [22] partitions the space into regions and maintains them in a Skip Graph using their split histories.

PRoBe is built on top of MURK as it uses a logical space for data distribution, routing and query answering. In contrast to MURK, it also proposes and evaluates techniques for load balancing and investigates the issue of using cached results. A number of techniques have been proposed for load balancing in DHTs [23, 24, 13], but they are not directly applicable to PRoBe since they rely on hashing and require a one dimensional identifier space. The idea of using cached results for efficient evaluation of future queries in a P2P environment have been explored in the context of range queries [25, 26, 27], OLAP queries [28], and conjunctive lookups [29].

Orthogonal to our work, P2P systems have also been used for more complex functionality such as sharing relational data and query processing [6, 7, 30, 31]. The adaptations of popular spatial index methods, such as R-trees and quad-trees, for a P2P setting have been devised [32, 33], but these methods still rely on certain statically fixed parameters.

### 3 The Design of PRoBe

To support range queries over multiple attributes, PRoBe organizes peers into a multi-dimensional logical space, similar to that used by CAN [9]. The dimensionality of this space is set to the number of *range attributes*, i.e., attributes over which range predicates can be specified<sup>1</sup>. Each dimension corresponds to an attribute and is bounded by the domain of the corresponding attribute.

The logical space is divided into non-overlapping rectangular regions called *zones*. Each peer in the system is assigned a zone and is responsible for maintaining the data items mapped to its zone. When a data item is inserted into the system, it is mapped to the point corresponding to its values for the range attributes. Figure 1(a) shows a 2-dimensional logical space corresponding to two range attributes:  $A_x$  with domain  $[100 - 200]$  and  $A_y$  with domain  $[0 - 80]$ . The space is partitioned among 7 peers,  $P_1$  to  $P_7$ . A data item with  $A_x = 150$  and  $A_y = 35$  is thus mapped to point  $P(150, 35)$  and assigned to peer  $P_4$ . When a new peer  $P_n$  joins the system, it contacts an existing peer  $P_e$  in the system.  $P_e$  then splits its zone into two in such a way that both halves contain the same number of data items and assigns one half to the new peer. In Fig.1(b), peer  $P_8$  joins the system and contacts peer  $P_4$ , which then splits its zone and hands over one half to  $P_8$ . For routing in the system, each peer keeps information (zone coordinates and IP addresses) about its neighbors in the logical space. During

<sup>1</sup> We assume that the number of range attributes is known at initialization time, or, in the worst case, all attributes can be considered range attributes.

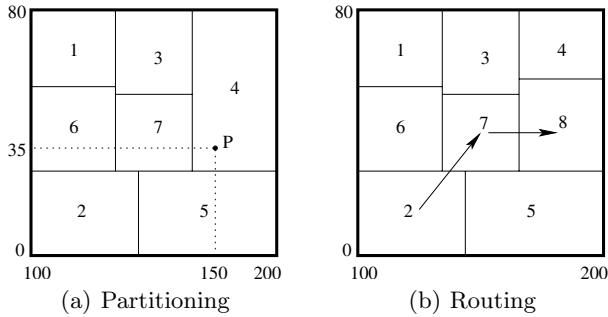


Fig. 1. Management of the Logical Space

routing, each intermediate hop forwards the message to the neighbor closest to the destination. Figure 1(b) shows the path for a message routed from  $P_2$  to  $P_8$ .

**Range Queries**

A range query specifies ranges over a subset of the range attributes. If it does not specify any range for a range attribute, then the domain of that attribute is used as the corresponding range predicate. Notice also that equality predicates are just a special case where the lower and upper bounds of a range are equal. Each range query then corresponds to a hyper-rectangle in the logical space, which will be referred to as the *query box*. Figure 2(a) illustrates two range queries  $Q_1$  and  $Q_2$ , where  $Q_1 = \{(116 < A_x < 160) \wedge (20 < A_y < 40)\}$  and  $Q_2 = \{A_x > 185\}$ .

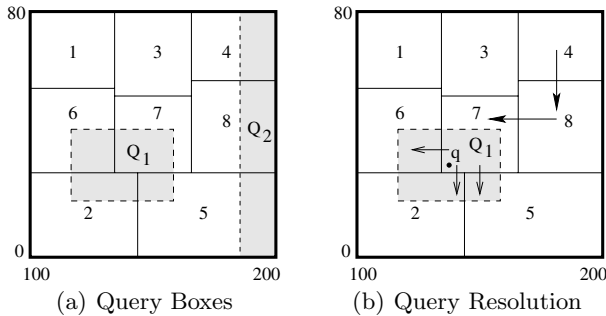


Fig. 2. Multi-Dimensional Range Queries

Since the data items and range queries are directly mapped onto the logical space, locality is preserved and the range queries can be answered by only visiting the peers with relevant data. All data items that are mapped to a point inside the query box satisfy the query and have to be returned. Thus answering a range query requires contacting all *candidate peers*, i.e., peers whose zones overlap with the query box. The query answering occurs in two phases in the following order:

1. **Routing:** The query is routed from the initiator to a peer whose zone intersects the query box.
2. **Retrieval:** The query is forwarded to all candidate peers, which in turn identify and return qualifying data items.

When routing the query, the initiator sets the destination to the center of the query box. Each peer that receives the query during routing checks if its zone overlaps with the query box. As soon as the query reaches a peer in the query box, the routing ends and that peer initiates the query retrieval. In the retrieval phase, each peer that receives the request passes over the data items it is assigned and returns those that satisfy the query to the query initiator. It also forwards the retrieve request to its neighbors whose zones intersect with the query box. Hence every peer in the query box will receive the query and all qualifying data items will be returned. In Fig.2, peer  $P_4$  initiates query  $Q_1$  and sends it towards the center of the query box, i.e., point  $q$ . When the query reaches peer  $P_7$ , which is the first peer whose zone overlaps with the query box, the retrieval phase begins and the query is forwarded to candidate peers  $P_2$ ,  $P_5$ , and  $P_6$ . These four peers then identify the data items matching the query and return them to peer  $P_4$ .

## 4 Improvements

### 4.1 Load Balancing

Most of the load balancing algorithms proposed for structured P2P systems cannot be used for P<sub>Ro</sub>Be because:

- They rely on hashing, but P<sub>Ro</sub>Be does not hash data,
- They require peers to hand over their partitions to their neighbors, however P<sub>Ro</sub>Be uses a multi-dimensional logical space so it is not always possible to find a neighbor that can be used for merging zones.

Thus we use a load balancing scheme based on *virtual servers* [23]. The basic idea is to allow a peer to manage multiple zones rather than a single zone. These zones are called virtual peers. The load balancing scheme works by allowing peers to start and stop virtual peers depending on the load distribution. When a peer  $P$  finds that the ratio between the load of the most loaded peer it knows of and its own load has crossed some fixed threshold  $T$ , it initiates the load redistribution process. First, peer  $P$  hands over its virtual peers to the peers responsible for the neighboring zones. The neighbor with the least load is used for handover. Then  $P$  splits the load with the most loaded peer such that after splitting their loads are nearly equal. This is achieved by transferring some virtual peers from the most loaded peer to  $P$  and if necessary splitting a virtual peer into two by dividing it across the current splitting dimension of the virtual peer. The splitting dimension for a virtual peer is chosen cyclically among all dimensions. A similar scheme is followed when the load of a peer is above  $T$  when compared to the least loaded peer.

A peer joins the system by splitting the load with the most loaded peer it can find. This splitting is similar to the load distribution process explained above. The load balancing scheme depends on a peer knowing the most loaded peer in the system. This can be achieved either by maintaining a separate 1-dimensional index on the load values [14, 12] or by polling random peers periodically about their load [13].

## 4.2 Sharing Cached Results

Caching is a widely used technique in databases for improving query processing performance. It is also used in P2P systems mainly for improving routing and lookup performance. For example, peers can cache the identity of other peers that are the recipient of messages or that return good lookup results. In PRoBe, we use caching for efficient evaluation of range queries. The idea is that if the answer of a range query is already computed and cached by a peer, then future range queries asking for similar ranges can benefit from that result. Note that the peers usually keep the answers of their range queries locally for a while. In such a data sharing environment, it is a realistic assumption that peers are also willing to share their cached results with others. These cached answers then can be used for efficiently evaluating future queries [25, 26, 27].

To facilitate the sharing and retrieval of cached results, PRoBe allows peers to share their cached results similar to the way they share data items. A cached result is mapped to the center point of the corresponding query box in the logical space. Thus whenever a peer wants to share its cached result for a given query, it sends an advertise message towards the center point of the query box and the peer responsible for that point then stores the identifying information, i.e., the range of the cached result and the address of the peer sharing it.

The query answering scheme presented in Sect.3 should be modified accordingly to consider cached results. A range query is still destined towards the center of the query box. However, now the routing does not stop when it reaches a zone intersecting the query box, instead the query is routed all the way to the destination peer  $P_d$  that is responsible for the center point.  $P_d$  then checks the advertised cached results it maintains and tries to locate a cached result that can be used for evaluating the query answer, i.e., a result that has a high overlap with the query range. If such a result is found,  $P_d$  contacts the caching peer so that it returns the matching portion of the cached result to the querying peer.  $P_d$  also generates sub-queries for the remaining parts of the query and forwards them to the corresponding peers. Thus the sub-queries are answered from the system, whereas the result for the rest of the query is obtained from the peer sharing the cached result.

This scheme uses the heuristic that if two queries are similar, then the centers of their query boxes will be close. However it is possible that the centers of two similar queries can map to different peers. The accuracy of the scheme can be increased by advertising the cached results for multiple points (e.g.,  $k$  different points on the diagonal of the query box) and having  $P_d$  also ask its immediate neighbors for overlapping cached results upon receiving a query. Additionally, in

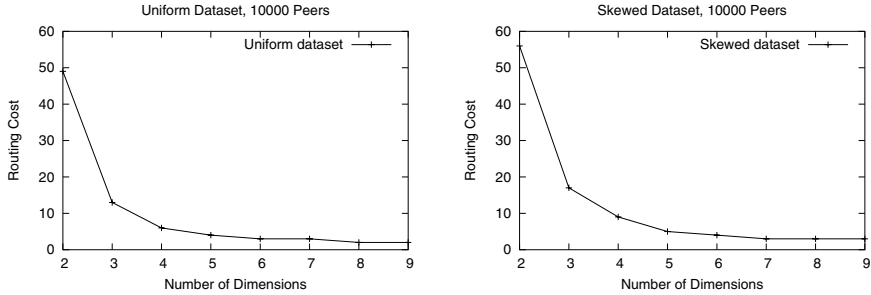
order to avoid stale results, caching peers periodically refresh their results and stop sharing a result after a certain time period. Thus the advertised cached results are removed from the system if they are not refreshed (for example when the caching peer leaves the system or stops sharing the result). Using a cached result is usually more efficient since all the results are obtained from a single peer. However the result might not be very accurate because there might have been matching data items inserted or removed after the cached result was computed. Thus the querying peer specifies in its query whether it is willing to accept cached results. For example, it might prefer not to use cached results if the accuracy of the query result is crucial.

## 5 Experiments

We implemented a simulator in C++ to understand the various aspects of PProBe. The domain of each attribute is set to  $[0, 10000)$ , thus the data space is a hyper-cube of side length 10000. We run each experiment using a uniform dataset and a skewed dataset, and report the results for both datasets. The uniform dataset is created by selecting data values uniformly at random from the corresponding attribute domain. In the case of skewed dataset, each attribute value is picked from a standard normal distribution, i.e., with mean 0 and standard deviation 1, and then scaled to the attribute domain as follows. It is mapped to the  $[-5, 5)$  interval using modular arithmetic, multiplied by 1000 and added 5000. The insertion of data objects and peers into the system are interleaved such that on the average one peer is inserted per 50 data objects. Queries are performed once all data and peer insertions are complete. All queries are hyper-cubes of specified coverage and they are uniformly spread over the data space. Query coverage is defined as the ratio of the volume of the query box to the volume of the data space. Each query is initiated at a random peer. We study different aspects of the system by varying three main system parameters, namely, the number of attributes (dimensions), query coverage, and the number of peers in the system. In the experiments where the number of dimensions is not varied, we use 2 and 4 dimensions as representative dimensions. Similarly coverages of  $10^{-5}$  and  $10^{-2}$  are used as representative query coverages. Unless otherwise stated, the number of peers in the system is 10000 and the load balancing algorithm is turned on. However caching is not used by default and is studied separately to isolate the effects of caching from other parameters of the system. For more efficient routing, whenever a peer has virtual peers, it considers the neighbors of all its zones to forward messages.

### 5.1 Response Time

Response time is the time interval between the time a range query was issued by a peer and the time the first result is received in response. We measure the number of hops taken by the query message from the querying peer to the first candidate peer and use it as an approximation for response time. This cost is



**Fig. 3.** Dimensionality vs. Routing Cost

referred to as the *routing cost*. Even though the number of hops may not always be an accurate measure in an Internet scale dynamic system, for the purpose of simulation and for understanding the nature of range query processing by the system it is a reasonable assumption.

Figure 3 shows that the routing cost decreases with increasing dimensions. That is because the number of neighbors per peer increases with the increased number of logical space dimensions, and thus there are more alternatives to forward the message at each intermediate hop. The routing is efficient for both datasets when the dimensionality is greater than 3. However the routing cost for lower dimensions (for 2 and 3 dimensions) is high. Therefore there is a need for an enhanced routing scheme for lower dimensions. A generalization of the long distance pointer scheme used in one dimensional schemes might mitigate this problem. For example, [20] suggests using random pointers and pointers based on building a one dimensional index on the centroids of zones to speed up routing. [34] discusses and evaluates different schemes for selecting long distance pointers. Similarly, [35] shows that it is possible to achieve  $O(\log N)$  routing performance in CAN by keeping neighbor pointers at different granularities (Each peer keeps  $O(\log N)$  routing information in this case).

While CAN divides the zones evenly between peers during joins, the expected routing cost is  $O(dN^{1/d})$ . In our case the partitioning is with respect to load and hence the data space is unevenly divided among the peers. We experimentally measure the impact of uneven partitioning on the routing cost by varying the number of peers. The results are shown in Fig.4. For 4 dimensions, the system scales well as the routing cost increases slowly with the increasing number of peers. However for 2 dimensions the rate of increase is higher. As already discussed, routing in 2 dimensions can be improved using long distance pointers.

As seen in Fig.3 and Fig. 4, the routing cost is higher for the skewed dataset. That is because there are more zones in the system due to our load balancing scheme, which creates virtual peers to cope with the data skew. The efficiency of the load balancing scheme will be investigated in Sect.5.4. For the skewed dataset in Fig.4, there is an unexpected increase in the routing cost in 2 dimensions around 8000 peers. That is because the number of virtual peers created in the

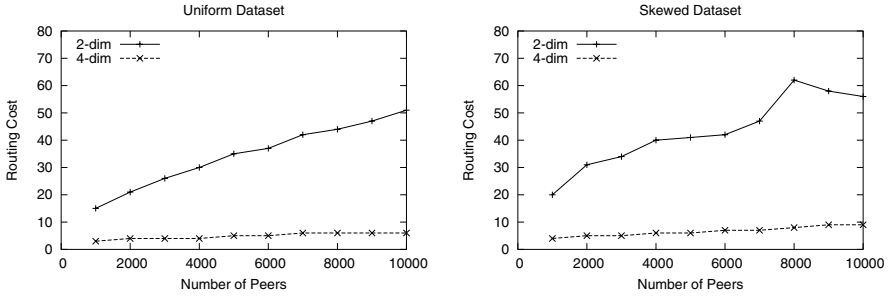


Fig. 4. Number of Peers vs. Routing Cost

case of 8000 was relatively fewer. Note that a peer uses the neighbors of all its virtual zones while deciding the next hop during routing for efficiency.

### 5.2 Retrieval Time

Retrieval time is the total time spent in the retrieval phase, which begins once the query reaches a candidate peer. We use the number of peers that are visited to retrieve the results as an approximation of the retrieval time. This is referred to as the *retrieval cost*. We first vary the number of dimensions from 2 to 9 and measure the change in the retrieval cost. The results are shown in Fig.5 for two different coverages of  $10^{-5}$  and  $10^{-2}$  (*y-axis is in log scale*). It can be seen that the space partitioning scheme is afflicted by the well-known curse of dimensionality. The performance degrades rapidly with increasing number of dimensions. The inefficiency is particularly acute for high coverage.

We also measure the effect of the number of peers and query coverage on the retrieval cost. Figure 6 plots the retrieval cost against the number of peers. The retrieval cost increases almost linearly with increasing number of peers. The effect of query coverage is shown in Fig.7. The performance is excellent for low coverages and is acceptable for coverages up to  $O(10^{-3})$ . Beyond this, performance rapidly degrades. This inefficiency is more pronounced in higher

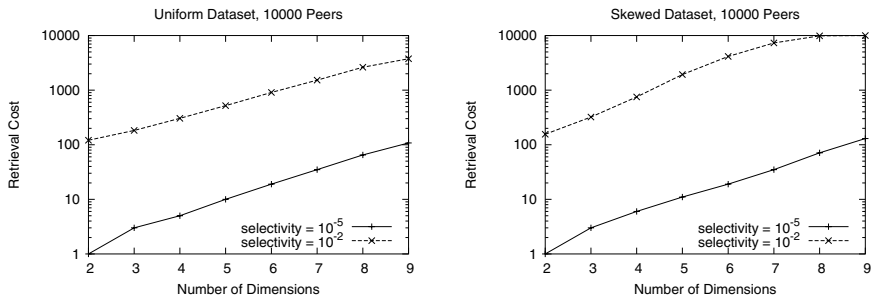
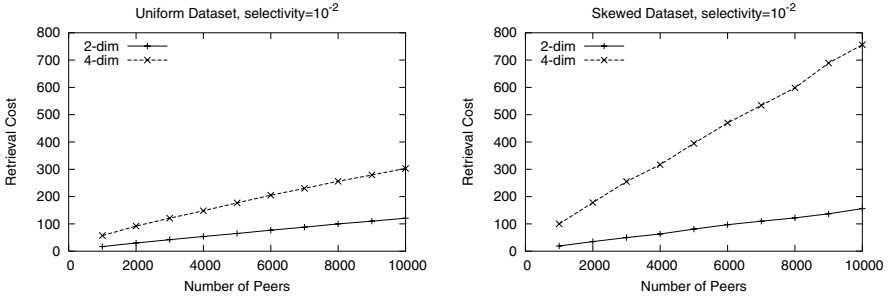
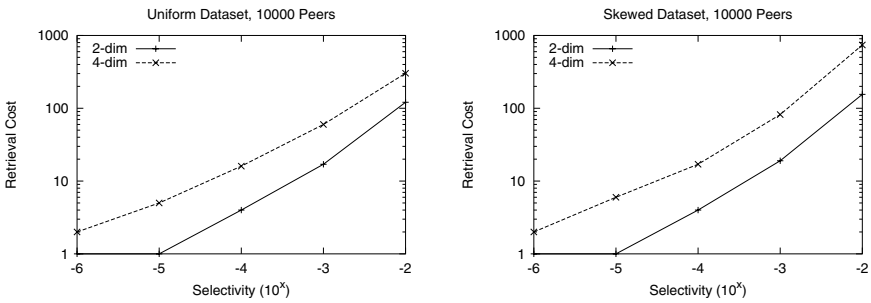


Fig. 5. Dimensionality vs. Retrieval Cost



**Fig. 6.** Number of Peers vs. Retrieval Cost



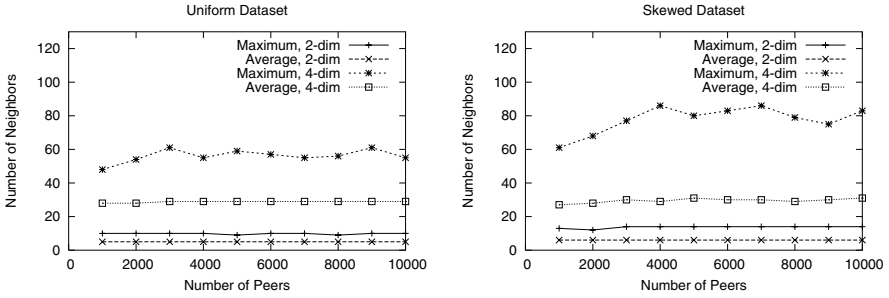
**Fig. 7.** Coverage vs. Retrieval Cost

dimensions which limits the utility of PRoBe for higher dimensions. However queries of high coverage can be efficiently cached as shown in Sect.5.5. Note that the query coverage affects only the retrieval cost since the routing scheme is independent of the query coverage.

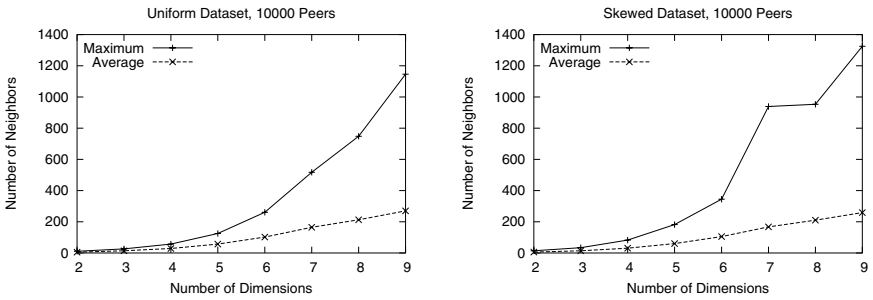
### 5.3 Maintenance Cost

In order to maintain the overlay network, a peer has to know all its neighbors and send heartbeat messages periodically. This is necessary for routing, recovering from peer failures, and processing range queries. Thus we use the maximum number of neighbors for any peer and the average number of neighbors over all peers as measures of the maintenance cost.

In Fig.8, the maintenance cost is plotted against the number of peers. For both datasets the average number of neighbors remain constant with increasing number of peers. However due to the uneven partitioning of the zones, the numbers are greater than the expected value  $O(2d)$  [9]. The number of neighbors for the skewed dataset is slightly higher than that of the uniform dataset. The maximum number of neighbors shows a slightly uneven behavior with increasing number of peers especially for the skewed dataset. This can also be explained by the uneven partitioning of the space.



**Fig. 8.** Number of Peers vs. Number of Neighbors



**Fig. 9.** Dimensionality vs. Number of Neighbors

Figure 9 depicts the number of neighbors as a function of the number of dimensions. The average number of neighbors per peer is low and increases rather slowly with increasing dimensions. However the maximum number of neighbors for the skewed dataset shows an uneven behavior with increasing dimensions. In spite of the uneven behavior two characteristics can be observed from the graphs for both datasets. The maximum number of neighbors is considerably higher than the average number of neighbors and the maximum number of neighbors degenerates to a very high cost for dimensions greater than 6. This is the price paid for the uneven partitioning of the space.

### 5.4 Load Balancing

The efficiency of the load balancing scheme is shown in Fig.10. The graph shows the corresponding number of peers for different values of storage load, i.e., the number of data objects assigned. In all runs, the load threshold is set to 4 and new peers join the system by splitting the highest loaded peer. Thus the load balancing scheme bounds the ratio of load between the maximum loaded peer to the minimum loaded peer to 4. In the case of uniform dataset, there are no virtual peers created. That is because the the load in the system is already balanced without the intervention of the dynamic load balancing scheme. As seen from

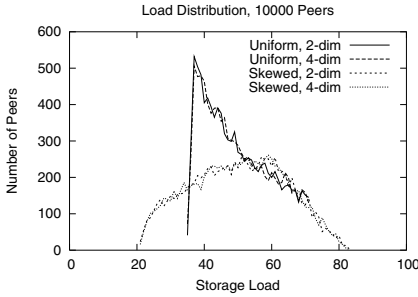


Fig. 10. Load Distribution

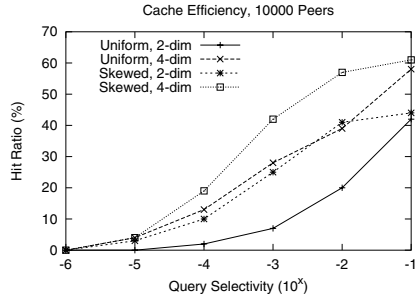


Fig. 11. Efficiency of Caching

the graph, most peers have load around 40 and the ratio of maximum load to minimum load is around 2. The dynamic scheme, however, effectively balances the load for the skewed dataset. In this case, there is one virtual peer per peer on the average, and the ratio of maximum load to minimum load is slightly less than 4 (For both dimensions, the maximum and minimum load are 83 and 21, respectively). Additionally, for both datasets, changing the dimensionality from 2 to 4 does not affect the load distribution.

### 5.5 Caching Efficiency

The efficiency of the caching protocol is shown in Fig.11. These results are obtained by running 10000 queries for different coverages. The  $y$  axis shows the percentage of the query volume covered by the local cached result that has the highest overlap with the query box. The percentage of results fetched from cache hits increases steadily with increasing query coverage. This can be readily explained since the amount of overlap between different queries is higher for high coverage queries. In all cases, the cache hit percentage rises to more than 40% for query coverage of  $10^{-1}$ . The results are higher for the skewed dataset due to large zones in sparse regions of the space. For a given dataset, the cache hit behavior with respect to increasing coverage is similar for 2 dimensions and 4 dimensions. Additionally, for any given coverage, the cache hit percentage is higher for 4 dimensions.

### 5.6 Discussion

From the experiments we conclude that:

1. PRoBe is best suited for low dimension and low coverage applications. For such applications, it provides excellent performance with low maintenance costs. It also scales well with increasing number of peers, which is an important criterion for dynamic P2P systems.
2. The routing performance is good for high dimensions. However a long distance pointer scheme is necessary for acceptable routing performance in low dimensions. The CAN-like routing alone is inefficient for low dimensions.

3. For high coverage queries in higher dimensions, P<sub>RO</sub>Be almost degrades to broadcasting among all peers (Figure 5). This suggests that for P2P systems for which these kind of queries are important a super-peer based broadcast architecture would be more efficient.
4. The above inference is further strengthened by high maintenance cost for higher dimensions. This can be reduced in exchange for less efficient performance by mapping higher dimensions to a lower dimensional logical space.
5. For high coverage queries, the base scheme is not very efficient. However when combined with caching, large portions of the query results can be answered from cached results of earlier queries. This is due to the high overlap between queries of high coverages.

## 6 Conclusions

In this paper, we have discussed the design and evaluation of P<sub>RO</sub>Be that efficiently supports range queries over multiple attributes in a P2P environment. P<sub>RO</sub>Be is based on a multi-dimensional logical space that is partitioned among participating peers, and also implements techniques for load balancing and sharing cached range query results.

There are several future directions we want to pursue. We plan to analyze how P<sub>RO</sub>Be compares to other schemes. We want to investigate routing improvements to accomplish more efficient routing in the system by keeping more routing information at peers. We will also explore the possibility of implementing a fully dynamic and decentralized tree-like index structure that efficiently supports range queries.

## References

1. Andrzejak, A., Xu, Z.: Scalable, efficient range queries for grid information services. In: P2P. (2002) 33–40
2. Cai, M., Frank, M., Chen, J., Szekely, P.: Maan: A multi-attribute addressable network for grid information services. In: GRID. (2003) 184–191
3. Gupta, A., Sahin, O.D., Agrawal, D., El Abbadi, A.: Meghdoot: Content-based publish/subscribe over p2p networks. In: Middleware. (2004) 254–273
4. Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM. (2004) 353–366
5. Tang, C., Xu, Z., Dwarkadas, S.: Peer-to-peer information retrieval using self-organizing semantic overlay networks. In: SIGCOMM. (2003) 175–186
6. Halevy, A.Y., Ives, Z.G., Suciu, D., Tatarinov, I.: Schema mediation in peer data management systems. In: ICDE. (2003) 505–516
7. Kementsietsidis, A., Arenas, M., Miller, R.J.: Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In: SIGMOD. (2003) 325–336
8. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: SIGCOMM. (2001) 149–160

9. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: SIGCOMM. (2001) 161–172
10. Zhao, B.Y., Huang, L., Stribling, J., Rhea, S.C., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A global-scale overlay for rapid service deployment. *IEEE Journal on Selected Areas in Communications* **22** (2004) 41–53
11. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: *Middleware*. (2001) 329–350
12. Ganesan, P., Bawa, M., Garcia-Molina, H.: Online balancing of range-partitioned data with applications to peer-to-peer systems. In: *VLDB*. (2004) 444–455
13. Karger, D.R., Ruhl, M.: Simple efficient load balancing algorithms for peer-to-peer systems. In: *SPAA*. (2004) 36–43
14. Crainiceanu, A., Linga, P., Machanavajjhala, A., Gehrke, J., Shanmugasundaram, J.: P-Ring: An index structure for peer-to-peer systems. Technical Report TR2004-1946, Cornell University (2004)
15. Aspnes, J., Shah, G.: Skip graphs. In: *SODA*. (2003) 384–393
16. Harvey, N.J.A., Jones, M.B., Saroiu, S., Theimer, M., Wolman, A.: SkipNet: A scalable overlay network with practical locality properties. In: *USITS*. (2003)
17. Daskos, A., Ghandeharizadeh, S., An, X.: PePeR: A distributed range addressing space for peer-to-peer systems. In: *DBISP2P*. (2003) 200–218
18. Awerbuch, B., Scheideler, C.: Peer-to-peer systems for prefix search. In: *PODC*. (2003) 123–132
19. Ramabhadran, S., Ratnasamy, S., Hellerstein, J.M., Shenker, S.: Brief announcement: prefix hash tree. In: *PODC*. (2004) 368–368
20. Ganesan, P., Yang, B., Garcia-Molina, H.: One torus to rule them all: multi-dimensional queries in p2p systems. In: *WebDB*. (2004) 19–24
21. Schmidt, C., Parashar, M.: Enabling flexible queries with guarantees in p2p systems. *Internet Computing Journal* **8** (2004) 19–26
22. Zhang, C., Krishnamurthy, A., Wang, R.Y.: SkipIndex: Towards a scalable peer-to-peer index service for high dimensional data. Technical Report TR-703-04, Princeton University (2004)
23. Rao, A., Lakshminarayanan, K., Surana, S., Karp, R., Stoica, I.: Load balancing in structured p2p systems. In: *IPTPS*. (2003) 68–79
24. Byers, J., Considine, J., Mitzenmacher, M.: Simple load balancing for distributed hash tables. In: *IPTPS*. (2003) 80–87
25. Gupta, A., Agrawal, D., El Abbadi, A.: Approximate range selection queries in peer-to-peer systems. In: *CIDR*. (2003) 141–151
26. Kothari, A., Agrawal, D., Gupta, A., Suri, S.: Range addressable network: A p2p cache architecture for data ranges. In: *P2P*. (2003) 14–22
27. Sahin, O.D., Gupta, A., Agrawal, D., El Abbadi, A.: A peer-to-peer framework for caching range queries. In: *ICDE*. (2004) 165–176
28. Kalnis, P., Ng, W.S., Ooi, B.C., Papadias, D., Tan, K.L.: An adaptive peer-to-peer network for distributed caching of OLAP results. In: *SIGMOD*. (2002) 25–36
29. Bhattacharjee, B., Chawathe, S., Gopalakrishnan, V., Keleher, P., Silaghi, B.: Efficient peer-to-peer searches using result-caching. In: *IPTPS*. (2003) 225–236
30. Huebsch, R., Hellerstein, J.M., Lanham, N., Loo, B.T., Shenker, S., Stoica, I.: Querying the Internet with PIER. In: *VLDB*. (2003) 321–332
31. Ng, W., Ooi, B., Tan, K., Zhou, A.: PeerDB: A p2p-based system for distributed data sharing. In: *ICDE*. (2003)

32. Mondal, A., Lifu, Y., Kitsuregawa, M.: P2PR-Tree: An R-Tree-based spatial index for peer-to-peer environments. In: P2P&DB. (2004) 516–525
33. Tanin, E., Harwood, A., Samet, H.: A distributed quadtree index for peer-to-peer settings (short paper). In: ICDE. (2005)
34. Sahin, O.D., Agrawal, D., El Abbadi, A.: Techniques for efficient routing and load balancing in content-addressable networks. In: P2P. (2005) 67–74
35. Xu, Z., Zhang, Z.: Building low-maintenance expressways for p2p systems. Technical Report HPL-2002-41, HP Laboratories Palo Alto (2002)