P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments

Anirban Mondal, Yi Lifu, and Masaru Kitsuregawa

Institute of Industrial Science,
University of Tokyo 4-6-1 Komaba, Meguro-ku, Tokyo, Japan
{anirban,yilifu,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. The unprecedented growth and increased importance of geographically distributed spatial data has created a strong need for efficient sharing of such data. Interestingly, the ever-increasing popularity of peer-to-peer (P2P) systems has opened exciting possibilities for such sharing. This motivates our investigation into spatial indexing in P2P systems. While much work has been done towards expediting search in file-sharing P2P systems, issues concerning spatial indexing in P2P systems are significantly more complicated due to overlaps between spatial objects and the complexity of spatial queries. Incidentally, existing R-tree-based structures for distributed environments (e.g., the MC-Rtree) are *not* adequate for addressing the sheer scale, dynamism and heterogeneity of P2P environments. Hence, we propose the P2PR-tree (Peer-to-Peer R-tree), which is a new spatial index specifically designed for P2P systems. The main features of P2PR-tree are two-fold. First, it is hierarchical and performs efficient pruning of the search space by maintaining minimal amount of information concerning peers that are far away and storing more information concerning nearby peers, thereby optimizing disk space usage. Second, it is completely decentralized, scalable and robust to peers joining/leaving the system. The results of our performance evaluation demonstrate that it is indeed practically feasible to share spatial data in a P2P system and that P2PR-tree is able to outperform MC-Rtree significantly.

Keywords: Spatial indexing, P2P systems, R-tree, scale, dynamism.

1 Introduction

Spatial data occurs in several important and diverse applications associated with geographic information systems (GIS), computer-aided design (CAD), resource management, development planning, emergency planning and scientific research. During the last decade, tremendous improvements in data gathering techniques have contributed to an unprecedented growth of available spatial data at geographically distributed locations and this coupled with the trend of increased globalization has created a strong motivation for the efficient global sharing of such data. Interestingly, the growing importance and ever-increasing popularity of peer-to-peer (P2P) systems such as Napster [10] and Kazaa [7], which have the capability of facilitating data sharing among hundreds of thousands of distributively-owned computers worldwide, has opened new and exciting possibilities for global sharing of spatial data. This motivates our investigation into spatial indexing

W. Lindner et al. (Eds.): EDBT 2004 Workshops, LNCS 3268, pp. 516–525, 2004.

in P2P systems. Interestingly, spatial data sharing in P2P systems can be of tremendous benefit to users looking for a hotel room or a museum or some landmark within a certain spatial location.

While much work has been done towards expediting search in file-sharing P2P systems [3, 11, 14], issues associated with the indexing of spatial data in P2P systems have received little attention. Understandably, several challenging issues such as overlaps between spatial objects, avoidance of data scattering and the complexity of spatial queries make the problem of spatial data indexing in P2P systems significantly more complicated than that of sharing files. Incidentally, spatial indexes have been extensively researched in centralized environments (e.g., the R-tree [6], the R*-tree [1] the R^+-tree [13] as well as in traditional distributed domains such as clusters (e.g., the dR-tree [9], the M-Rtree [8] and the MC-Rtree [12]). Existing R-tree-based techniques use centralized mechanisms in which a centralized 'master' node supervises and directs queries to all other nodes in the system. However, we believe that such centralization is *not* appropriate for P2P systems partly due to the fact that all updates and searches passing through a centralized peer may result in severe performance problems (and even more so if the centralized peer goes offline[1]) and partly due to the fact that it is practically extremely challenging to maintain updated information about other peers at a centralized peer when data can be added/deleted autonomously by any peer in the entire system or peers can join/leave the system anytime. In essence, a completely decentralized spatial indexing technique, which is scalable enough to handle hundreds of thousands of peers and also dynamic enough to deal with peers joining/leaving the system anytime, is called for.

This paper proposes the P2PR-tree (Peer-to-Peer R-tree), which is a new spatial index specifically designed for P2P systems. The main features of P2PR-tree are two-fold.

1. It is hierarchical and performs efficient pruning of the search space by maintaining minimal amount of information concerning peers that are far away and storing more information concerning nearby peers, thereby optimizing disk space usage.
2. It is completely decentralized, scalable and robust to peers joining/leaving the system, thereby making it well-suited to P2P environments.

We conducted simulation experiments to test the effectiveness of P2PR-tree for spatial select (window) queries. The results indicate that it is indeed practically feasible to share spatial data in a P2P system and that P2PR-tree is able to outperform MC-Rtree significantly. The remainder of this paper is organized as follows. Section 2 discusses related work, while Section 3 discusses the problem context as well as our proposed scheme for spatial indexing in P2P environments. Section 4 reports our performance evaluation. Finally, we conclude in Section 5 with directions for future work.

2 Related Work

The problem of spatial indexing has motivated several research efforts. In this regard, the R-tree [6] is one of the most popular spatial index structures. Each spatial data object

[1] Irrespective of how a centralized peer may be selected, no guarantee can be provided about the peer remaining online all the time.

in the R-tree is represented by a Minimum Bounding Rectangle (MBR). Leaf nodes in the R-tree contain entries of the form *(oid, rect)* where *oid* is a pointer to the object in the database and *rect* is the MBR of the object. Non-leaf nodes contain entries of the form *(ptr, rect)* where *ptr* is a pointer to a child node in the R-tree and *rect* is the MBR that covers *all* the MBRs in the child node. Variants of the R-tree include the R^+-tree [13] and the R*-tree [1].

R-tree-based distributed spatial indexes include the M-Rtree [8], MC-Rtree [12] and the dR-tree [9]. In case of the M-Rtree, all the internal nodes of the parallel R-tree are stored at one single dedicated machine that is regarded as the master server, while the leaf nodes are declustered across several other machines. The leaf level at the master server contains the (MBR, siteid, pageid) tuples for each global leaf node. In order to identify the page and site where the leaf page is located, the (siteid, page id) is used. An improvement to the M-Rtree is the MC-Rtree where the master node contains *only* the client ids of the data nodes (and *not* page ids), while the data rectangles are kept indexed by R-trees in the client machines. Intuitively, MC-Rtree exploits parallelism better than the M-Rtree since the client machines find the page ids in parallel. The dR-tree uses an R-tree-based two-tier indexing mechanism which facilitates efficient data migration and load-balancing in clusters. More recently, an R-tree-based indexing structure for P2P systems has been proposed in [5] in the context of sensor networks. The proposed index structure in [5] can be interpreted as a P2P version of the centralized R-tree. However, our work differs from the proposal in [5] in several ways. Here, we state only two of the major differences. First, our approach is completely decentralized without any notion of cluster leaders, while the work in [5] assumes the existence of cluster leaders. Second, the execution of nearest neighbour queries have been optimized in [5], while we focus on optimizing window queries.

3 Distributed Spatial Indexing in P2P Environments

This section first discusses the context of the problem and then proposes the *P2PR-tree* for efficiently locating objects in spatial P2P environments.

Problem Context

Given a set of hundreds of thousands of geographically distributed and distributively owned data providing peers, the problem is to search efficiently for a given spatial object such that the queried object can be retrieved within response times that would be acceptable to most users.

Every peer has a globally unique identifier *peer_id* (when a peer leaves the system and then joins the system, the ID remains preserved.) We need to adopt any existing identification scheme used for P2P systems. Every peer stores data concerning certain spatial regions. Note that spatial attributes usually remain relatively static, but non-spatial attributes may change e.g., a hotel's geographical location can be reasonably expected to remain the same over a significantly long period of time, but the number of available rooms in the hotel can change very frequently. Moreover, every incoming query is assigned a unique identifier *Query_id* by the peer P_i at which it arrives. *Query_id*

consists of *peer_id* and *tm,* where *tm* is a distinct number generated by P_i using the time of arrival of the query as a seed. Observe that this ensures uniqueness of *Query_id* since more than one query cannot arrive at the same peer at *exactly* the same time. We define a peer P_i as *relevant* to a query Q if it contains at least a non-empty subset of the answers to Q, otherwise P_i is regarded as *irrelevant* w.r.t. Q. Note that it is possible for more than one peer to store information concerning the same spatial region and possibly even the same spatial objects. Moreover, it is *not* necessary that each peer indexes all the spatial objects that are within the covering MBR of the region that it indexes. This may be primarily attributed to the fact that the owner of each peer autonomously decides the spatial objects about which he wishes to store information. We shall henceforth designate the covering MBR of the region indexed by a peer as the *peerMBR* of that peer.

The P2PR-Tree

In case of P2PR-tree, the universe is first divided *statically* into a set of *blocks* (each block being a rectangular tile). The set of blocks will constitute level 0 of our proposed index as we shall see later. Each block is *statically* divided into a set of *groups* (each group is a rectangular tile) and the set of groups constitute level 1 of our index. Each group is *dynamically* divided into further rectangular tiles and these set of tiles, designated as *subgroups,* forms level 2 of our index. Depending upon the circumstances, subgroups may be *dynamically* divided further into sets of rectangular tiles, which we shall designate as subgroups of level 3. Note that we shall use the term *subgroups* generically throughout this paper to indicate sets of rectangular tiles which form level i of our index, where $i \geq 2$.

The static decomposition of space has an important advantage from the perspective of P2P systems. Whenever a new peer joins the system, it just needs to contact one peer which will inform it about the covering MBRs of the blocks and *at least one* peer in each block. Using this block structure information, the peer can decide which block(s) it belongs to. (In case the region indexed by a peer overlaps more than one block, the peer will be assigned to both blocks.) Once the peer knows its block(s), it contacts one peer inside its block for the group-related MBR information and at least one peer inside each group. Using the group structure information, the peer will know which group it belongs to. Once the peer assigns itself to that group, it finds out which subgroup it should assign itself to and so on. Note that this strategy optimizes disk space usage significantly by maintaining minimal information about peers that are far away and more detailed information concerning peers that are nearby. Interestingly, this kind of static decomposition of space is able to deal efficiently with peers joining and leaving the system. On the other hand, if we had dynamically divided the universe into blocks, information about any splits in blocks would have to be sent to an extremely large number of peers. Moreover, in case of a dynamic way of dividing the universe into blocks and groups, it would have been extremely challenging to keep the block-related and group-related information updated, given the dynamic nature of P2P systems. However, the dynamic method of splitting is an attractive option when the number of peers is low since the dynamic method can deal with highly skewed data distributions which the static technique cannot. Hence, for levels other than block and group levels, we perform dynamic decomposition of space, as we shall see shortly. Notably, the maximum

number of peers in a group is pre-specified at design time and we shall denote it by G_{Max}. Moreover, the maximum number of peers in subgroups (at different levels of the distributed index) is also specified at design time and we shall refer to it as SG_{Max}.

Assume the universe is divided into 4 blocks, namely b1,b2,b3,b4. Now let us take a closer look at a specific block, say b1, to understand detailed issues concerning how P2PR-tree works. Figure 1a depicts the distribution of peerMBRs in each of the 4 groups (namely g1,g2,g3,g4) of b1, while Figure 1b presents the corresponding index structure. In Figure 1, P1,P2,P3,P4, P5,P6, P8,P9,P10 denote the peerMBRs of peers whose *peer-ids* are 1,2,3,4,5,6, 8,9,10 respectively. In Figure 1b, B1,B2,B3,B4 represent the covering MBRs of b1,b2,b3,b4 respectively, while G1,G2,G3,G4 represent the covering MBRs of g1,g2,g3,g4 respectively. For the sake of clarity, we display the index structure with special emphasis *only* on G1 and G2. Observe that the number of peerMBRs in

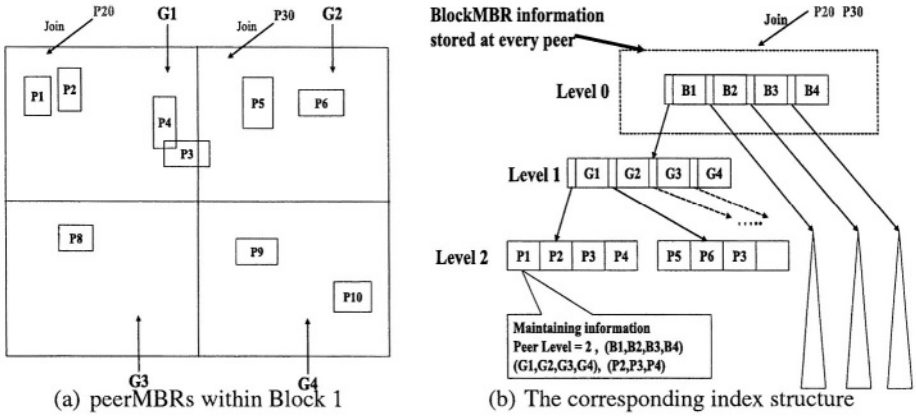


Fig. 1. Illustrative example depicting our indexing scheme

each group is *not* the same e.g., while G1 has 4 peerMBRs, G3 has only 1 peerMBR. This kind of skew occurs primarily because the static decomposition of space is *not* based upon the actual data distribution during run-time. Given that peers may join/leave the system at any time, the number of peerMBRs corresponding to a given group can be reasonably expected to keep changing dynamically, the implication being that skews among groups are inevitable because it is *not* feasible to have a priori knowledge concerning the dynamically changing data distributions in each of the groups. Similarly, a moment's thought indicates that such data skews may also occur at any other level of our distributed index. Interestingly, it is also possible for a peerMBR to overlap multiple groups e.g., P3 overlaps both G1 as well as G2. In case a peerMBR overlaps more than one group, the corresponding peer will be assigned to both the groups. Hence, in the index structure shown in Figure 1b, P3 appears *twice* since P3 has been assigned to both G1 and G2. Note that we define P_L, the level of a peer, based upon its position in the distributed index e.g., P_L in case of peer 6 is 2 in Figure 1b.

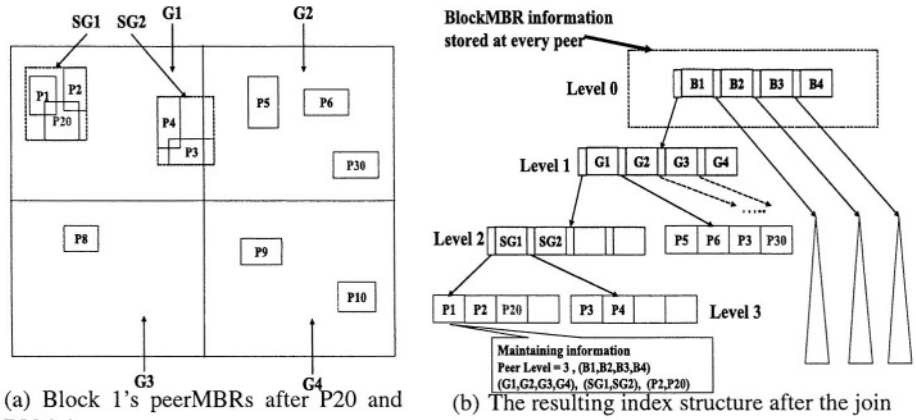


Fig. 2. Example to show the effect of peers joining

Now let us understand how the index is modified in response to new peers joining the system. Figure 2a depicts what happens when two new peers join the system with their respective peerMBRS P20 and P30. For this example, let us assume the values of G_{Max} and SG_{Max} to be 4. Observe that P30's joining the system is straightforward since it does *not* result in an overflow. However, P20's joining is significantly more complicated since its joining causes an overflow in G1, thereby causing G1 to split further into subgroups SG1 and SG2. For splitting purposes, we propose to adopt an existing clustering technique [2] for performing node splitting in R-trees. The main idea behind the proposal in [2] is that the node splitting problem in R-trees is essentially a problem of finding a good set of clusters and the proposal also moves beyond the traditional two-way node splitting of R-trees to make node splitting more flexible, the prime objective being to find *real* clusters as opposed to two groupings.

Observe that the node splitting caused P1 and P2 to move from level 2 to level 3 of the index. From Figure 2b, it is clear that P2PR-tree does *not* provide global height-balance. In Figure 1b, we have shown the information that P1 maintains to facilitate search. As we see from Figure 1b, P1 maintains information concerning the entire covering MBRs of each of the blocks, namely B1 ,B2,B3,B4 and the covering MBRs of all the 4 groups in its own block (i.e., G1,G2,G3,G4) in addition to the peerMBRs P2,P3,P4. Observe from Figure 2b how the information maintained by P1 is changed after splitting occurs.

Search Mechanism

Now we shall discuss how efficient search can be conducted via P2PR-tree. For our search mechanism, every query is associated with a Q_L, the significance of Q_L being that it determines which level of the distributed index the query is currently traversing. When a new query is issued to any peer in the system, its Q_L is 0 and whenever a query is forwarded to peer(s) at another level of the distributed index, the value of Q_L is incremented by one. This guarantees that queries traverse down the distributed index

and precludes the possibility of any query traversing up the index. Whenever a query Q arrives at any peer P_i in the system, P_i checks whether its peerMBR intersects with Q and if so, P_i searches its own R-tree, returns results (if any) and the search is terminated. Otherwise, P_i checks the value of Q_L associated with Q and depending upon the value of Q_L, P_i forwards Q to the relevant block(s) or group(s) or subgroup(s) or peer(s) as the case may be. (If Q_L is 0, Q should be forwarded to the relevant block(s); if Q_L is 1, Q should be sent to relevant group(s); if Q_L is 2, Q must be sent to relevant subgroup(s) and so on.)

P_i sending Q to a particular block B_i constitutes Q being sent to one peer in that block. Note that this implicitly assumes that every peer knows at least one peer in each block. While the system is operational and the peers issue queries to each other, it is likely that more peers will interact and come to know each other. Hence, over a period of time, it might be possible for a peer in a specific block to know N peers in each of the other blocks. Given that P_i knows N peers at block B_i to which it wishes to forward Q, P_i first sends Q randomly to any one peer P_j among the N peers that it knows. If it does *not* receive an acknowledgement message from P_j within a pre-specified maximum time limit, designated as TIME_OUT, P_i forwards the query to another peer among the N peers that it knows. In case all the N peers that P_i knows in B_i are unavailable, P_i will *not* be able to forward Q to B_i. For the sake of convenience, we shall henceforth refer to the set of N peers that a peer knows in each block (or in each group/subgroup) as the *routing peers* or simply *routers.* Note that the mechanisms for sending a query to a particular group or subgroup are essentially similar to that of sending a query to a specific block.

4 Performance Study

We conducted simulation experiments to evaluate the performance of our proposed indexing strategy. Our simulation environment comprised a machine running the Solaris 8 operating system. The machine has 4 CPUs, each of which has a processing power of 900 MHz. Main memory size of the machine is 16 Gigabytes, while the total disk space is 2 Terabytes. For all our experiments, we divided the universe into 10 blocks and we divided each block into 10 groups. Moreover, we set the value of TIME_OUT to 20 seconds. Transfer time between peers (inter-block) was randomly varied between 80 ms to 120 ms, while the transfer time between peers (inter-group) was varied between 45 ms to 55 ms. Transfer time between peers within the same group/subgroup was randomly varied between 10 ms to 15 ms. Note that an interarrival rate of n queries/second implies that n queries were issued in the entire P2P system every second. By availability of $x\%$, we mean that at any given time, a peer has an $x/100$ probability of being online (available). Furthermore, the number of *routers* being set to y means that each peer knows y peers (for routing purposes, hence we designate them as *routers*) in each block in the system and y routers in each group of its own block and y routers in each subgroup of its own group and so on.

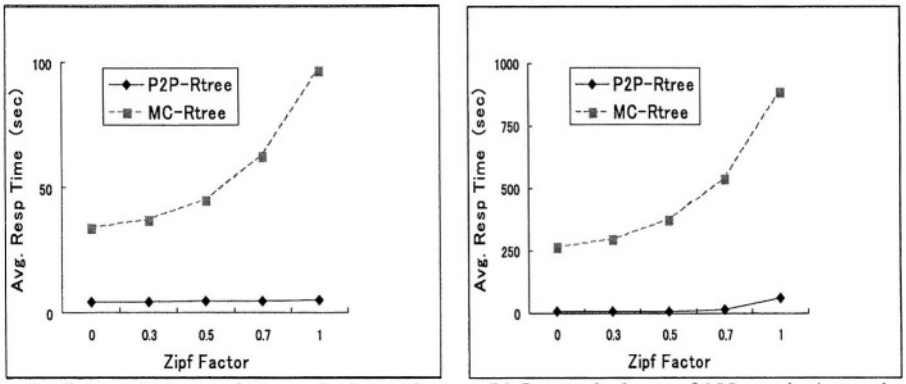
Each of the 1000 peers that we used in all our experiments stored more than 200000 spatial objects. Each peer uses an R-tree for its own directory management. As in existing works, we assumed that one R-tree node fits in a disk page (page size = 4096 bytes).

The height of each of the R-trees at each of the 1000 data providing peers was 3 and the fan-out was 64. Our performance study was conducted using a real dataset known as *Greece Roads* [4]. We had enlarged this dataset by translating and mapping the data. The main metric that we have used for the performance study is query response time.

Incidentally, existing works on spatial indexing have *not* really addressed issues concerning P2P environments, let alone decentralized indexing techniques. In order to compare our work meaningfully against existing works, we use MC-Rtree as reference. Recall that MC-Rtree is one of the most efficient distributed R-tree-based techniques. (We do *not* compare our approach with the M-Rtree since the MC-Rtree has been shown to outperform the M-Rtree.) For the MC-Rtree approach, we select a specific peer in every block as the block leader. Each of these block leaders maintains an MC-Rtree which indexes the peerMBRs of all the peers whose spatial regions are fully contained within their blocks or intersect with their blocks. We ensured that every block leader had adequate disk space for storing the MC-Rtree. For the sake of convenience, we shall henceforth refer to this strategy as *MC-Rtree.*

Now let us study the effect of varying the zipf factor when the query interarrival rate is fixed. In order to model skewed workloads, we used the Zipf distribution over 1000 buckets to decide the number of queries to be directed to each of the 1000 peers. We modified the value of the *zipf factor* to obtain variations in workload skew. Notably, a value of 1 for the zipf factor implies a heavily skewed workload, while a value of 0 indicates a uniform workload distribution. We generated window queries by enlarging the individual data MBRs at each of the peers. Figure 3a indicates the results when the query interarrival rate was fixed at 20 queries/second, while the results for query interarrival rate of 100 queries/second is shown in Figure 3b. The number of routers was set to 5 and the availability was fixed at 65%.

From the results in Figure 3a, we find that as the skew increases, the average response time also increases for the MC-Rtree. This occurs because *every* query has to be routed through *at least one* of the centralized master peers which store the MC-Rtree for their respective blocks. As a result, there are large job queues at these centralized master peers, thereby causing significantly increased waiting times at these peers which ultimately causes severely increased query response times. The greater the workload skew, the more serious is the routing bottleneck. In contrast, the decentralized nature of P2PR-tree implies that routing is performed in a completely distributed fashion, thereby ensuring the absence of any serious routing bottlenecks. This explains why P2PR-tree exhibits far superior performance as compared to MC-Rtree. The same explanation is also applicable to the results in Figure 3b. Observe that in Figure 3b, the actual values of response times are much higher than in Figure 3a. This is because high interarrival rates make the routing bottleneck associated with the centralized master peers much more pronounced than in case of low interarrival rates. Incidentally, apart from the routing bottleneck, MC-Rtree also needs to contend with individual peers becoming bottlenecks due to a large number of queries being directed to a few 'hot' peers (because of the highly skewed workload) within a short time interval. Interestingly, the phenomenon of individual peers becoming bottlenecks due to skewed workload at high interarrival rates also occurs in case of P2PR-tree which explains why unlike the results in Figure 3a, the results in



(a) Interarrival rate of 20 queries/second (b) Interarrival rate of 100 queries/second

Fig. 3. Effect of variations in workload skew

Figure 3b indicate an increase (albeit slight) in the average response time for P2PR-tree when the interarrival rate is high (i.e., 100 queries/second).

5 Concluding Remarks

The increased importance of geographically distributed spatial data coupled with the popularity of P2P computing has motivated our research into spatial indexing in P2P systems. Since existing R-tree-based structures are *not* adequate for P2P environments, we have proposed a new R-tree-based spatial index that is well-suited to P2P environments. Our performance evaluation demonstrates that it is indeed practically feasible to share spatial data in a P2P system. However, this work has *not* addressed in detail issues concerning a single peer indexing multiple regions that are far apart in space. Moreover, this work does *not* examine performance-related issues concerning a single query intersecting more than one block/group/subgroup. We intend to investigate these issues in detail in the near future. We also wish to make detailed performance comparisons with the proposal in [5]. Furthermore, we intend to investigate issues concerning replication for performance as well as availability reasons and additionally, we plan to examine issues concerning load-balancing in this context for improving user response times.

References

1. N. Beckmann, H.P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *Proc. ACM SIGMOD,* 1990.
2. S. Brakatsoulas, D.Pfoser, and Y. Theodoridis. Revisiting R-tree construction principles. *citeseer.nj.nec.com/586207.html.*
3. A. Crespo and H. G. Molina. Routing indices for Peer-to-Peer systems. *Proc. ICDCS,* 2002.
4. Datasets. http://dias.cti.gr/~ytheod/research/datasets/spatial.html.
5. M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. *Proc. P2P,* 2003.

6. A. Guttman. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD,* 1984.
7. Kazaa. http://www.kazaa.com/.
8. N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. *Proc. EDBT,* 1996.
9. A. Mondal, M. Kitsuregawa, B.C. Ooi, and K.L. Tan. R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases. *Proc. ACM GIS,* 2001.
10. Napster, http://www.napster.com/.
11. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM,* 2001.
12. B. Schnitzer and S.T. Leutenegger. Master-client R-trees: A new parallel R-tree architecture. *Technical Report COMP-98-01, University ofDenver,* 1998.
13. T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. *Proc. VLDB,* 1987.
14. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. ACM SIGCOMM,* 2001.