On Spatial Indexing in Peer-to-Peer Environments

Yilifu Anirban Mondal Masaru Kitsuregawa
Institute of Industrial Science
University of Tokyo, Japan
{yilifu,anirban,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 among users. 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 P2PR-tree (Peer-to-Peer R-tree), which is a new R-tree-based indexing mechanism specifically 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 the MC-Rtree significantly.

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

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 among users. Interestingly, the growing importance and ever-increasing popularity of peer-to-peer (P2P) systems such as Napster[13] and Kazaa[10], 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 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, 15, 19], 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 makes the problem of spatial data indexing in P2P systems significantly more complicated than that of sharing files. In this regard, DHT(Distributed Hash Table) based P2P systems[19], are *not* adequate for indexing spatial objects primarily because they may potentially result in scattering the spatial data over a large number of peers, thereby resulting in considerable overlap and consequently causing even small window queries to access a large number of peers. However, we believe that the time has come to investigate the possibility of deploying P2P systems for sharing spatial data globally.

Incidentally, spatial indexes have been extensively researched in centralized environments (e.g., the R-tree[8], the R*-tree[1], the R^+-tree[18]) as well as in traditional distributed domains such as clusters (e.g., the dR-tree [12], the M-Rtree[11] and the MC-Rtree [16, 17]). However, existing R-tree-based techniques for distributed environments are *not* adequate for P2P environments for two reasons. First, peers can join/leave anytime in P2P systems and the existing techniques have *not* been designed to deal with this kind of dynamism. Second, the existing techniques use centralized mechanisms comprising a centralized node which 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 because it is practically extremely challenging to maintain updated information at a centralized peer when data can be added/deleted autonomously by any peer in the entire system. 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.

Notably, important ongoing grid-related projects such as the GRID Physics Network (GriPhyN)[14] and the European DataGRID[4] have the capability of sharing data which is expected to be in the multi-terabyte range. Our proposal differs from these works mainly in two ways. First, individual nodes in spatial GRIDs are usually dedicated and they may be expected to be available most of the time, while for P2P systems, nodes may join or leave arbitrarily at any point of time. Second, given that computers in spatial GRIDs are owned by organizations, we can have considerable amount of centralized control in GRIDs. In contrast, the individual peers in P2P systems are usually distributively owned, thereby precluding the possibility of any kind of centralized control.

This paper proposes the P2PR-tree (Peer-to-Peer R-tree) scheme for supporting efficient spatial indexing in P2P systems. The main features of P2PR-tree are two-fold.

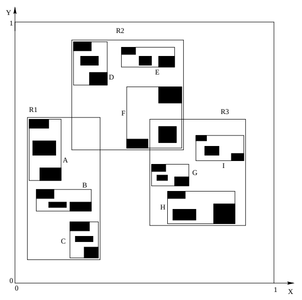
1. It is hierarchical and performs efficient pruning of the search space by maintaining minimal amount of information concerning peers that are far away and storing more information concerning nearby peers, thereby optimizing disk space usage.

2. It is completely decentralized, scalable and robust to peers joining/leaving the system, thereby making it well-suited to P2P environments.

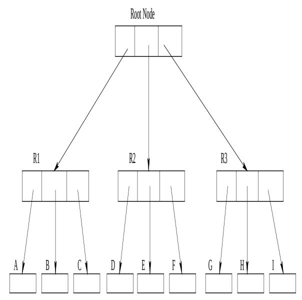
We conducted simulation experiments to test the effectiveness of P2PR-tree for spatial select (window) queries. The results indicate that it is indeed practically feasible to share spatial data in a P2P system and that P2PR-tree is able to outperform the MC-Rtree significantly. The remainder of this paper is organized as follows. Section 2 discusses related work, while Section 3 presents an overview of our proposed system. Section 4 discusses our proposed scheme for spatial indexing in P2P environments, while Section 5 reports our performance evaluation. Finally, we conclude in Section 6 with directions for future work.

2. RELATED WORK

The problem of spatial indexing has motivated several research efforts. In this regard, the R-tree [8] is one of the most popular and widely used spatial index structures. Each spatial data object 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. Figure 1a depicts a set of data rectangles organized in an R-tree, while Figure 1b indicates how they are indexed by an R-tree with fanout 3. The bounding rectangles at each level of the R-tree



(a) Data rectangles organized in an R-tree (data rectangles are shaded in black)



(b) The corresponding R-tree

Figure 1: The R-tree

can overlap and this may possibly lead to an R-tree search traversing multiple paths down the tree, thereby resulting in increased number of disk accesses . With the objective of reducing such overlaps, some variants of the R-tree, such as the R$^+$-tree [18] and the R*-tree [1] have been proposed. While the R$^+$-tree avoids overlapping rectangles in the intermediate nodes of the tree, the R*-tree gives preference to squarish covering boxes with the objective of reducing overlaps.

R-tree-based distributed spatial indexes include the M-Rtree [11], MC-Rtree [16] and the dR-tree [12]. 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.

Ongoing research efforts aimed at global sharing of spatial data are essentially GRID-related and include the GRID Physics Network (GriPhyN)[14], the European DataGRID[4], the Earth Systems GRID (ESG)[7] and the NASA Information Power GRID (IPG)[9].While the GriPhyN project and the European DataGrid project both aim at employing GRIDS for improving scientific research which require efficient distributed handling of data in the petabyte range, the ESG project aims at facilitating detailed analysis of huge amounts of climate data by a geographically distributed

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

community via high bandwidth networks. The IPG project attempts to improve existing systems in NASA for solving complex scientific problems efficiently. More recently, an R-tree-based indexing structure for P2P systems has been proposed in [6] in the context of sensor networks. The proposed index structure in [6] can be interpreted as a P2P version of the centralized R-tree. However, our work differs from the proposal in [6] 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 [6] assumes the existence of cluster leaders. Second, the execution of nearest neighbour queries have been optimized in [6], while we focus on optimizing window queries.

3. PROBLEM OVERVIEW

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.

Deciding upon the amount of information that a peer must maintain about the data stored at other peers in the system represents a trade-off between information maintenance cost (number of messages and index maintenance overhead) and search efficiency (response time and number of messages) . If a peer maintains *no* information concerning other peers, search has to be performed by broadcast which results in flooding the network with queries and possibly increased user response times. On the other hand, if a peer maintains full information concerning other peers, search efficiency may be increased significantly albeit at the cost of high maintenance. Moreover, the information may not fit in main memory, thereby necessitating disk I/Os to access the information. More importantly, given that peers may join/leave the system at any time and a very large number of data items may be added or updated or deleted within a very short time interval, it is *not* a practically viable option for a peer to maintain full information about other peers. Keeping this in mind, our approach maintains some information about other peers to facilitate indexing. However, in this paper, we do *not* specifically investigate the optimal granularity at which a peer should maintain information about other peers and hence, we leave granularity issues to further study.

4. A DISTRIBUTED SPATIAL INDEXING SCHEME FOR P2P ENVIRONMENTS

This section presents our proposed *P2PR-tree (Peer-to-Peer R-tree)* spatial indexing scheme for efficiently locating objects in spatial P2P environments. In case of P2PR-tree, the

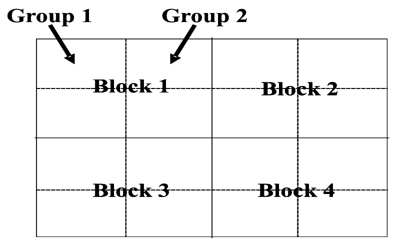
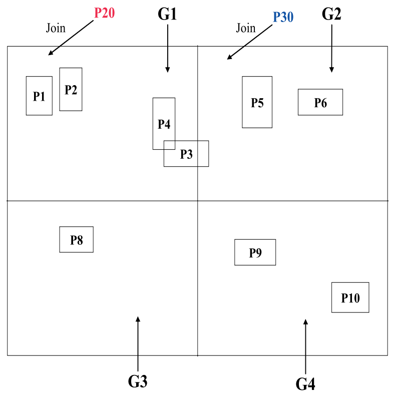


Figure 2: Creating a hierarchical static decomposition of space

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.

The static decomposition of space has an important advantage from the perspective of P2P systems. Whenever a new peer joins the system, the newcomer 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 spaces 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



(a) peerMBRs within Block 1

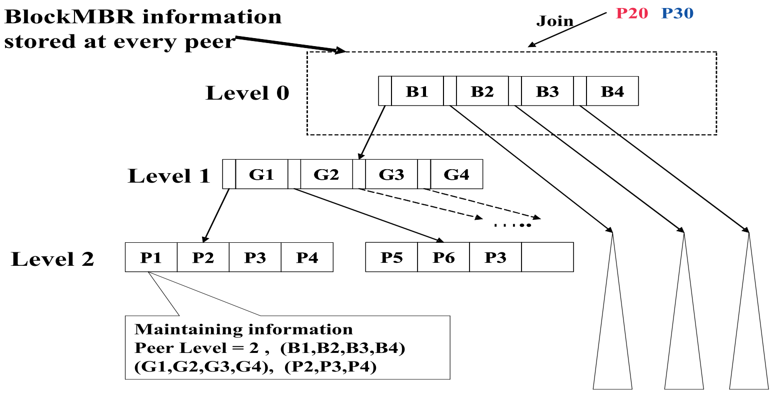
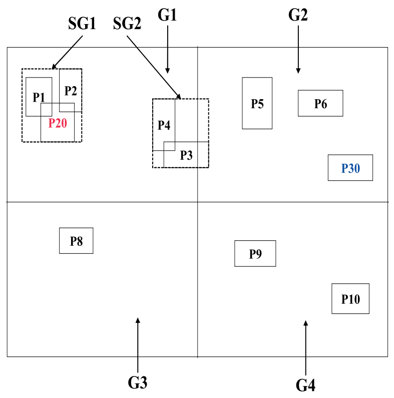

BlockMBR information stored at every peer
Join
P20 P30
Level 0
B1
B2
B3
B4
Level 1
G1
G2
G3
G4
Level 2
P1
P2
P3
P4
P5
P6
P3
Maintaining information
Peer Level = 2 , (B1,B2,B3,B4)
(G1,G2,G3,G4), (P2,P3,P4)
(b) The corresponding index structure

Figure 3: Illustrative example depicting our indexing scheme



(a) Block 1's peerMBRs after P20 and P30 joins

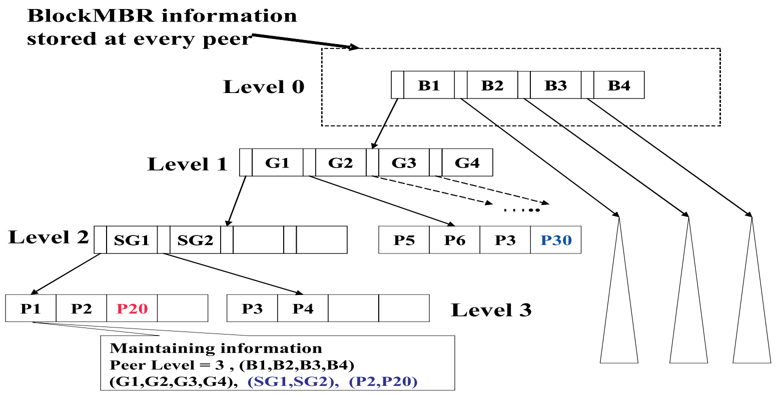

BlockMBR information stored at every peer
Level 0
B1
B2
B3
B4
Level 1
G1
G2
G3
G4
Level 2
SG1
SG2
P5
P6
P3
P30
P1
P2
P20
P3
P4
Level 3
Maintaining information
Peer Level = 3 , (B1,B2,B3,B4)
(G1,G2,G3,G4), (SG1,SG2), (P2,P20)
(b) The resulting index structure after the join

Figure 4: Example to show the effect of peers joining

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. 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 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}.

Figure 2 indicates how the universe is divided into 4 blocks (Block1, Block2, Block3 and Block4) and groups. Now let us take a closer look at block 1 to understand detailed issues concerning how P2PR-tree works. Figure 3a depicts the distribution of peerMBRs in each of the 4 groups (namely G1,G2,G3,G4) of block 1, while Figure 3b presents the corresponding index structure. In Figure 3, P1,P2,P3,P4, P5,P6, P8,P9,P10 denote the peerMBRs of peers whose *peer_id*s are 1,2,3,4,5,6, 8,9,10 respectively. In Figure 3b, B1,B2,B3,B4 represent the covering MBRs of block1,block2,block3 and block4 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 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 3b, 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 3b.

Now let us understand how the index is modified in response to new peers joining the system using the P2PR-tree scheme. Figure 4a 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 4b, it is clear that P2PR-tree does *not* provide global height-balance. In Figure 3b, we have shown the information that P1 maintains to facilitate search. As we see from Figure 3b, 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 of P2,P3,P4. Observe from Figure 4b how the information maintained by P1 is changed after splitting occurs.

Notably, the fact that our approach is R-tree-based (since we use MBRs as an approximation) does not restrict the peers in the system to using only R-tree-based structures for indexing the spatial objects that they store. A peer can use *any* indexing scheme for indexing its objects, but the only requirement of P2PR-tree is that the peer must be able to provide its peerMBR to other peers in the system.

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). Figure 5 depicts our proposed search algorithm. 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 a 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

```
Algorithm SpatialP2PSearch( )
Input: Query MBR $Q$
Output: Query Result if result is found, NULL otherwise
MBR_Intersect ( ): A function which returns TRUE if two
MBRs intersect, FALSE otherwise

/* A query $Q$ is issued to a peer $P_i$ */

if ( MBR_Intersect ($Q$, $PeerMBR_{P_i}$ ) {
  Search own R-tree
  terminate
} else {
  Check the value of $Q_L$
  if ( $Q_L$ = = 0 ) {
    Check level 0 to decide relevant block(s)
    $Q_L$ ++;
    Send $Q$ to each relevant block
  } else if ( $Q_L$ = = 1 ) {
          Check level 1 to decide relevant groups(s)
          $Q_L$ ++;
          Send $Q$ to each relevant group
  } else {
          if ($P_L > Q_L$) {
            Check level $Q_L$ to decide relevant subgroups
            $Q_L$ ++;
            Send $Q$ to each relevant subgroup
          } else if ( $P_L$ = = $Q_L$ ) {
            Check level $Q_L$ to decide relevant peers
            Send $Q$ to relevant peer(s)
          }
  }
}
end
```

Figure 5: Spatial P2P search algorithm

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 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.

5. PERFORMANCE STUDY

This section reports the performance evaluation of our proposed technique.

Experimental setup

We conducted extensive 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.

The main metric that we have used for the performance study is query response time since we believe that response time provides a reasonably accurate reflection of search performance. We also measured the number of queries that failed due to peers being unavailable (offline). In particular, our performance evaluation investigates the following:

- Effect of variations in interarrival rate of queries
- Effect of workload skew
- Effect of variations in the number of routers per peer

Table 1 provides a summary of the parameters that we used in our performance study. For all our experiments, we divided the universe into 10 blocks and we divided each block into 10 groups. In Table 1, TIB denotes transfer time between peers (inter-block), while the transfer time between peers (inter-group) is represented as TIG. TIP is the transfer time between peers within the same group, while InterRate denotes the interarrival rate between queries. For example, when InterRate is 10 queries/second/peer, it implies that 10 queries are issued to every peer in the system every second. Moreover, we set the value of TIME_OUT to 20 seconds.

Each of the 1000 peers that we used in 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. Given that the number of free-riders in P2P systems is typically much higher than that of data providing peers, we also took free-riders into consideration for our simulation study. In particular, for our simulation, we assume that the free-riders do *not* have any index structure stored at themselves, thereby implying that a free-rider first has to contact any one of the data providing peers in the entire system and this data providing peer will use the P2PR-tree scheme to process the query. Our performance study was conducted using a real dataset known as *Railroads in Germany*[5]. We had enlarged this dataset by translating and mapping the data.

Parameter	Default Value	Variations
No. of Peers	1000	5000
Zipf factor	0	0.1,0.3,...,0.9
InterRate(queries/sec/peer)	20,40,...,100	
TIB (ms)	80 to 120	
TIG (ms)	45 to 55	
TIP (ms)	10 to 15	
Availability(percentage)	80	20,40,60,100
No. of Routers	5	1,2,3,4

Table 1: Parameters used in Performance Study

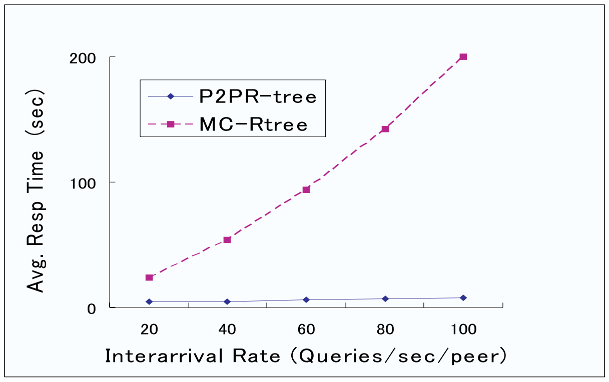
In order to model skewed workloads, we used the Zipf distribution over n buckets to decide the number of queries to be directed to each of the n peers in the system. Note that this is only an approximate manner of generating skewed workloads since the actual load imposed on a peer depends not only upon the number of queries directed to that peer, but also on the individual *sizes* of the respective queries. Moreover, due to overlaps in spatial regions between peers, it cannot be guaranteed that a specific query would only be relevant to one peer. This is essentially an inevitable compromise. 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 stored at each of the peers.

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 the MC-Rtree as reference. Recall that the MC-Rtree is a well-known and 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 R-tree. For the sake of convenience, we shall henceforth refer to this strategy as the *MC-Rtree*.

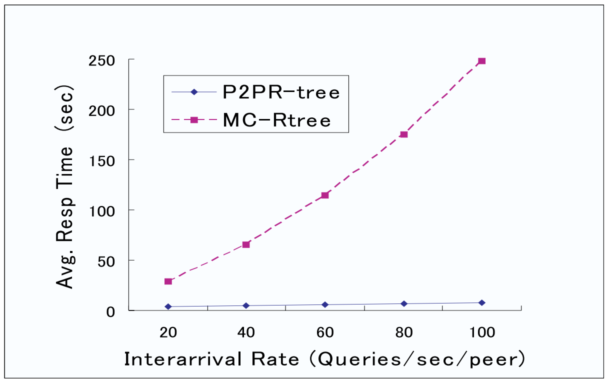
Variation in query interarrival rate

Now let us investigate the comparative performance of P2PR-tree with respect to MC-Rtree when the query interarrival rate is varied, while keeping the zipf factor fixed at 0. For this experiment, the number of routers was set to 5 and the availability of every peer was fixed at 80%. Figure 6a and Figure 6b depict the results of the effect of variations in query interarrival rates when the number of peers in the system was 1000 and 5000 respectively.

The results in Figure 6 indicate that with increase in query interarrival rate, the response time increases only slightly in case of P2PR-tree, while for MC-Rtree, the response



(a) Effect of varying interarrival rates for 1000 peers



(b) Effect of varying interarrival rates for 5000 peers

Figure 6: Performance comparison between P2PR-tree and MC-Rtree

time keeps increasing. Such increased response times occur in case of MC-Rtree because *every* query has to be routed through at least one of the centralized block leaders, thereby resulting in a large number of queries waiting for long periods of time to be routed and ultimately causing severely increased query response times. In contrast, the completely decentralized nature of P2PR-tree ensures the absence of any serious routing bottlenecks, thereby explaining why P2PR-tree exhibits superior performance as compared to MC-Rtree.

Effect of variations in the workload skew

Now we shall study the effect of variations in the workload skew by varying the zipf factor, the number of routers being set to 5 and the availability being fixed at 80%. The query interarrival rate was fixed at 10,000 queries/second for the whole system i.e., 10000 queries come to the system every second and the number of queries to be directed to each peer depends upon the zipf factor. The results, which are presented in Figure 7, demonstrate that the performance gain of P2PR-tree over MC-Rtree keeps increasing as the workload skew increases. Observe that as the workload skew increases, there is a slight increase in response time for P2PR-tree primarily because highly skewed workloads cause some of the peers to become bottlenecks, thereby resulting in increased response times at those peers. However, the increase in response time for MC-Rtree (as the skew increases) is much greater than that of P2P-Rtree because apart from individual 'hot' peers, the MC-Rtree strategy also has to contend with serious routing bottlenecks and load imbalance at the centralized block leaders. Interestingly, the results also demonstrate that even though the decentralized nature of P2PR-tree ensures the absence of bottlenecks in routing, large number of queries directed to a few 'hot' peers within a small time interval can still degrade the performance of P2PR-tree to some extent. One way of preventing such degradation in performance is to integrate load-balancing techniques into P2PR-tree. Hence, we intend to examine load-balancing issues in this context in the near future.

As the results in Figure 6 and Figure 7 demonstrate, the performance of P2PR-tree is far superior to that of MC-Rtree. Hence, we shall not discuss or make performance comparisons with MC-Rtree any further. Now let us study the performance-related behaviour of P2PR-tree in response to variations in the number of routers known to a peer.

Variation in the number of routers

The number of routers that each peer knows in each block can impact system performance considerably. Hence, we performed an experiment in which we varied the number of routers that a peer knows per block, while keeping the zipf factor fixed at 0 and the interarrival rate set to 10 queries/second/peer. The experimental results in Figure 8 indicate that when the availability of individual peers is high, the success rate of queries is high and depends little on the number of routers per peer e.g., in Figure 8, when all the peers are available 100% of the time, the success rate is 100% irrespective of the number of routers. However as the availabilities of individual peers decrease, the effect of the number of routers that a peers knows becomes more and more pronounced. For example, for 1 router per peer and 20% availability of individual peers, the success rate of queries falls below 20%. The results indicate that the success rate of queries increases with an increase in the number of routers per peer. This is expected because the more the number of routers that a peer knows, the higher would be the probability of success for a given query.

6. CONCLUDING REMARKS

The unprecedented growth and increased importance of geographically distributed spatial data has created a strong need for efficient sharing of such data among users. Interestingly, the growing importance and ever-increasing popularity of peer-to-peer (P2P) systems have opened new and exciting possibilities for global sharing of spatial data. This provides a strong motivation for designing a spatial P2P system which allows its users *transparent* access to data of *any* location from *anywhere*. While much work has been done for expediting search in file-sharing P2P systems, issues associated with search in geo-spatial P2P systems are significantly more complicated due to overlaps between spatial objects and the complexity of spatial queries. Incidentally, R-tree-based structures, which have been proposed in

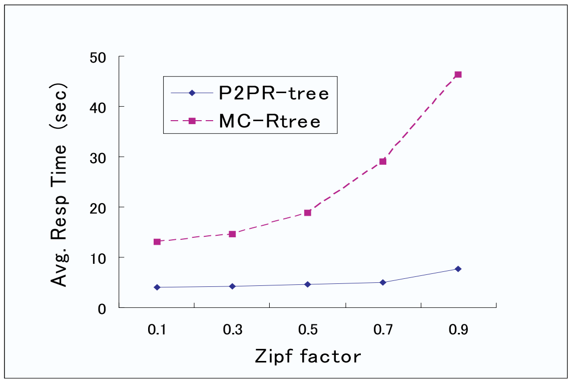


Figure 7: Effect of varying zipf factor

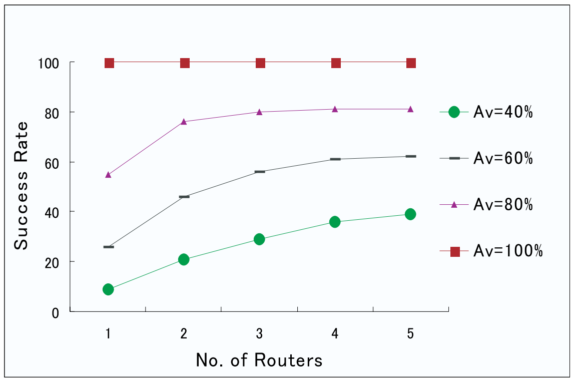


Figure 8: Effect of varying the number of routers

existing works concerning indexing of spatial data in distributed environments (e.g., clusters), are *not* adequate for addressing the sheer scale, dynamism (in particular, peers joining/leaving the system and new data being added or deleted from the peers anytime) and heterogeneity of P2P environments. Hence, we have proposed a new R-tree-based mechanism, which is specifically designed for efficiently indexing spatial objects in a P2P environment. The proposed technique is completely decentralized, scalable and robust to peers joining/leaving the system. Moreover, the technique optimizes disk space usage by maintaining minimal amount of information concerning peers that are far away and storing more information concerning nearby peers and its hierarchical nature ensures efficient pruning of the search space. We conducted extensive simulation experiments to test the effectiveness of our proposed spatial indexing technique for spatial select (window) queries. Our performance evaluation demonstrates that the response times for user queries are within acceptable limits and that it is indeed practically feasible to share spatial data in a P2P system.

To this end, we believe that our contributions have addressed some of the relevant issues associated with spatial indexing in P2P systems. Currently, we are working on algorithms for handling highly skewed data distributions. In the near future, we intend to investigate issues concerning replication for performance as well as availability reasons and also we plan to examine issues concerning load-balancing in this context with the objective of improving user response times.

7. 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. *In 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] European DataGrid. http://eu-datagrid.web.cern.ch/eu-datagrid/.

[5] Datasets. http://dias.cti.gr/∼ytheod/research/datasets/spatial.html.

[6] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. *Proc. P2P*, 2003.

[7] Earth Systems Grid. http://www.earthsystemgrid.org/.

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching. *In Proc. ACM SIGMOD*, pages 47–57, 1984.

[9] NASA IPG. http://www.ipg.nasa.gov/.

[10] Kazaa. http://www.kazaa.com/.

[11] N. Koudas, C. Faloutsos, and I. Kamel. Declustering spatial databases on a multi-computer architecture. *In Proc. EDBT*, pages 592–614, 1996.

[12] 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.

[13] Napster. http://www.napster.com/.

[14] GriPhyN Project. http://www.griphyn.org/index.php.

[15] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM*, 2001.

[16] B. Schnitzer and S.T. Leutenegger. Master-client R-trees: A new parallel R-tree architecture. *Technical Report COMP-98-01, University of Denver; URL: http://www.cs.du.edu/ leut/ssdbm99-TR.ps*, 1998.

[17] B. Schnitzer and S.T. Leutenegger. Master-client R-trees: A new parallel R-tree architecture. *In Proc. of Statistical and Scientific Database Management*, pages 68–77, 1999.

[18] T. K. Sellis, N. Roussopoulos, and C. Faloutsos. The R^+-tree: A dynamic index for multi-dimensional objects. *In Proc. VLDB*, pages 507–518, 1987.

[19] 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.