ASPEN: An Adaptive Spatial Peer-to-Peer Network*

Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku
Computer Science Department, University of Southern California
Los Angeles, CA, USA
haojunwa, rzimmerm, wku@usc.edu

ABSTRACT

Geographic Information Systems (GIS) are increasingly managing very large sets of data and hence a centralized data repository may not always provide the most scalable solution. Here we introduce a novel approach to manage spatial data by leveraging structured Peer-to-Peer (P2P) systems based on Distributed Hash Tables (DHTs). DHT algorithms provide efficient exact-match object search capabilities without requiring global indexing and as a result they are extremely scalable. Furthermore, the adoption of uniform hash functions ensures excellent load balancing. However, range queries – which are very common with spatial data – cannot be executed efficiently because the hash functions unfortunately destroy any existing data locality. Here we report on the design of an Adaptive Spatial Peer-to-pEer Network (ASPEN) that extends Content Addressable Networks (CAN) to preserve spatial locality information while also retaining many of the load balancing properties of DHT systems. We introduce the concept of *scatter regions*, which are spatial data distribution units that optimize both load balancing and spatial range query processing at the same time. We present a data object key generation function and algorithms for spatial range queries. We rigorously evaluate our technique with both synthetic and real world data sets and the results demonstrate the efficient execution of spatial range queries in the ASPEN system.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*distributed databases*; H.2.8 [**Database Management**]: Database Application—*spatial databases and GIS*

General Terms

Algorithms, Design, Performance

*This research has been funded in part by NSF ITR grant CMS-0219463, equipment gifts from Intel and Hewlett-Packard, unrestricted cash grants from the Lord Foundation and by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152.

GIS'05, November 4–5, 2005, Bremen, Germany.

Keywords

Peer-to-Peer Systems, Distributed Spatial Databases

1. INTRODUCTION

Recent research work in the area of structured Peer-to-Peer (P2P) systems has produced novel approaches for the scalable distribution of large data sets in a decentralized manner. As a common characteristic, in systems such as CAN [10], Chord [13], and Pastry [11], node identifiers are usually randomly assigned when nodes join the systems. Each node only needs to maintain a small routing table with a few neighbor nodes entries. These systems use Distributed Hash Tables (DHTs) to allocate data objects to nodes with no central control. This method ensures the uniform distribution of data objects, which results in superb scalability, load balancing and robustness of DHT systems. With DHT *exact match* queries can be performed in a very efficient way. In a n-node system, the process of locating a particular data object on average requires only O(log n) search steps.

Some of the large scale data sets now available are distributed in nature. For example, Geographic Information Systems (GIS) utilize spatial data, such as road networks, maps, and boreholes, from various data sources, integrating them with textual and other information and creating innovative and powerful end user applications. By their very nature, queries on spatial data commonly concern a certain range or area, for example queries relate to intersections, containment, and nearest neighbors. Thus the spatial locality of data objects needs to be preserved when spatial data objects are distributed. However, DHT mechanisms adopted by most structured P2P systems tend to distribute data objects uniformly and maintain no spatial locality. Consequently, spatial queries in these systems can only be supported inefficiently through an exhaustive search. On the other hand, spatial data objects are usually not uniformly distributed within their space domain (e.g., road networks are more dense in cities). If we simply select a locality-preserving hash function, the uniform data distribution will be destroyed, which may results in a performance bottleneck. Therefore, it is desirable to scatter data objects in the areas with a high object density to achieve a certain level of load balancing.

In this paper we propose an Adaptive Spatial Peer-to-pEer Network (ASPEN), which is based on our previous work [16], to store spatial data objects in distributed environments with constrained load balancing and the preservation of spatial locality. Our work leverages the design of CAN [10] to support spatial data management in P2P systems. The contributions of our work are:

- We introduce the concept of *scatter regions*. A scatter region is a logical data distribution unit that covers multiple zones in a physical plane. Instead of uniformly hashing data

objects within the complete space, our design introduces a data object key generation function to uniformly distribute data objects within their scatter region. This function is able to efficiently support spatial range queries, as well as exact match queries.

- We describe a system design for distributing spatial data objects with fixed (static) scatter regions first. The proposed approach constrains the data distribution to provide data locality while retaining load balancing properties. However, P2P systems are dynamic and self-organizing by nature, hence the number of nodes and the data object distribution vary over time. Consequently, we extend our design to provide dynamically adjustable scatter regions, which autonomously adapt their size with regard to changes in the node and data object distribution.

- We propose a spatial range query algorithm in ASPEN. Our complexity analysis and simulation results indicate that the hop count and the message count for our spatial range query algorithm are of order O($n^{1/d}$) and O(n), respectively, in a n node system with d dimensional, equally partitioned space.

- We report extensive experimental results based on two spatial data sets. Our results demonstrate that the ASPEN system is able to provide constrained load balancing with a wide variety of scatter region sizes.

The rest of this paper is organized as follows. The related work is described in Section 2. In Section 3 we introduce ASPEN's scatter region design, the data object key generation function, and algorithms in support of spatial range queries. The experimental validation of ASPEN, using both synthetic data and real world data, is presented in Section 4. Finally, we discuss the conclusions and future work in Section 5.

2. RELATED WORK

A number of studies have addressed range query support in DHT systems [5], [7], [12]. Gao et al. [5] introduced the Range Search Tree (RST) to map data partitions into nodes. Range queries are decomposed into sub-queries corresponding to nodes in the RST trees. Hellerstein et al. [7] use prefix hash trees, which hashes common prefixes into a multi-way retrieval tree, to support range queries. Schmidt et al. [12] used the Hilbert Space-Filling Curve [3] in the index space to support partial keywords and range queries in DHT systems. However, these techniques are focused on one dimensional data, thus spatial data cannot be directly supported by these approaches.

Load balancing in DHT systems has been studied in [6] and [2]. Aspnes et al. [2] proposed a mechanism to assign elements with similar keys to the same server to maintain load balancing in the system. Rao et al. [6] introduced the concept of *virtual server* in their design. Each virtual server is assigned the same load. Nodes are dynamically grouped so that each virtual server remains below its load capacity. However, these techniques focus only on one dimensional data with no consideration to spatial locality preservation.

Recent work that addressed spatial data support in DHT systems are [9] and [14]. Tanin et al. [14] proposed a design for efficiently querying spatial data objects over structured P2P systems. Based on the Quad-tree algorithm [4], their design recursively divides a two dimensional (2-D) space into smaller grid areas. Each grid space is managed by a control point. Each data object is associated with the smallest grid that contains the object in its entirety.

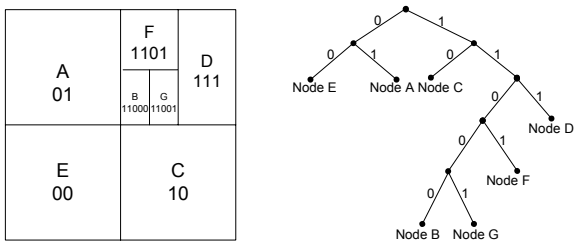


Figure 1: An example of CAN plane and its Virtual Identifier (VID) tree.

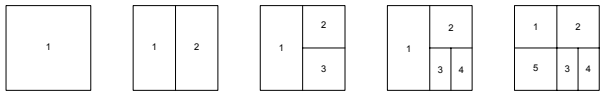


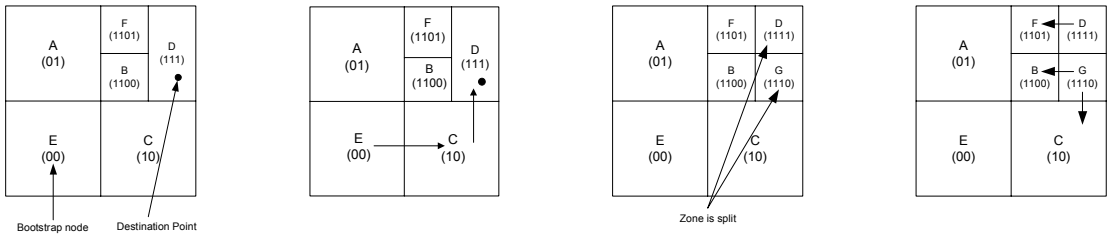
Figure 2: Example of CAN space partitioning as 5 nodes join in succession in 2-D dimension.

The 2-D space is thus transformed into a tree structure that stores spatial data objects. Each control point is hashed to a node in the underlying Chord ring [13]. Although this design can preserve spatial locality by directly mapping control points to nodes in Chord, the system may not always achieve good load balancing: if a grid area contains abundant data objects, a single control point will be responsible for all of them. Similarly, Mondal et al. [9] proposed the design of a P2PR-tree, a decentralized R-tree based index for spatial data in DHT systems. Their design combines static space decomposition at the top index levels with dynamic space grouping at the lower levels of indexing to cope with highly skewed data distribution over peers. However, peers generally have a high frequency of joins and departures in P2P systems, it is extremely challenging to keep the index updated with an R-tree based dynamic space grouping algorithm. Although this design addresses the issue of handling highly skewed data, more efficient algorithms are desirable to achieve good load balancing in spatial DHT systems.

Content-addressable networks (CAN) introduce a logical Cartesian coordinate plane to store data objects. We leverage the design of CAN by applying physical spatial meaning to the coordinate plane for storing spatial data objects. The next section reviews the basic operation of CAN.

2.1 Content-addressable Networks (CAN)

CAN introduces a novel approach for creating a scalable indexing mechanism in P2P environments. It creates a logical d-dimensional Cartesian coordinate plane divided into *zones*, where zones can be dynamically partitioned or merged as nodes join and depart. Each zone in this plane is addressed with a Virtual Identifier (VID), which is calculated from the location of a zone in the logical plane. Each node in the system is responsible for storing all data objects assigned to a specific zone. Figure 1a shows an example of a 2-D plane partitioned into 7 CAN zones. Each zone is controlled by a node and the node address is represented by its VID. The corresponding structure of the VID tree is shown in Figure 1b. The concatenation of branch values on the path from the tree root node to a specific node represents its VID. The partitioning is performed by following a well-known ordering of dimensions in deciding along which dimension the partition is to be done. For example, in a 2 dimensional space, the partitioning first occurs along the X dimension, then the Y dimension, then X again followed by Y, and so forth. Figure 2 shows the partitions of a 2-D CAN space as five nodes join in succession.



(a) Finding a bootstrap node

(b) Routing to the destination zone

(c)Zone space split

(d) Routing information update

Figure 3: The node insertion operation in CAN.

CAN nodes operate without global knowledge of the plane. Each node maintains a routing table that consists of the IP addresses and logical zone areas of its immediate neighbors. For each CAN message with a destination coordinate, nodes consult their routing table and simply apply a greedy forwarding algorithm to route the message to the neighbor with coordinates closest to the destination location.

When a new node joins a CAN system, several steps must be taken to allocate a zone for it. First, the new node must find an existing node (also called the bootstrap node), which is already a member of the CAN system. Second, the bootstrap node randomly chooses a destination coordinate in the logical CAN plane, and the bootstrap node routes the new node to the zone covering that destination coordinate. Third, the destination zone is split into halves, each being controlled by one node. Finally, the neighbors of the split zone will be notified to update their routing tables.

Figure 3 illustrates the node insertion operation in CAN. When a new node G joins the system, it first uses node E as the bootstrap node. Node E generates a destination point in the plane, which is in the zone controlled by node D in this example (Figure 3(a)). Node E routes node G to node D through node C (Figure 3(b)). A greedy algorithm that selects the neighbor with coordinates closest to the destination location is applied in the routing. Then the destination zone is split into two halves, which are controlled by nodes D and G, respectively. Figure 3(c) shows the CAN plane after the node insertion. Note that the VID of node D changes in this operation. The routing table of nodes D and G and their neighbors (nodes B, C, and F in this example) also need to be updated (Figure 3(d)).

When a data object is about to be inserted or removed, CAN generates a key based on the object identifier (e.g., filename) and inserts the data object as a {key, value} pair. The key is mapped into a point P in the CAN plane by using a uniform hash function. The {key, value} pair is inserted or removed at the node which controls the point P. For the retrieval of a data object, the same hash function is applied to the key in order to regenerate the point P in the logical CAN plane. The query is routed to the zone which controls the point P, to retrieve the data object.

CAN is designed to uniformly distribute data objects in distributed environments with no spatial locality preservation. Spatial queries thus can be inefficiently supported by an exhaustive search. However, CAN constructs a pure logical coordinate plane to uniformly distribute data objects. This characteristic allows us to extend the logical plane with physical spatial meaning for distributing spatial data objects. We describe the design of *scatter regions* in Section 3, which constrain spatial data object distribution in ASPEN to achieve both good load balancing and spatial locality preservation.

3. ASPEN SYSTEM DESIGN AND COMPONENTS

In this section we describe the core components of the ASPEN system. In Section 3.1 we introduce the concept of spatial data distribution units, called *scatter regions*. Next, in Section 3.2, we present a novel spatial data object key generation function that assigns data objects to scatter regions. We address the ASPEN system operations and introduce a spatial range query algorithm with static scatter regions in Section 3.3. Finally, we extend our design to dynamic scatter regions in Section 3.4. For a concise presentation, we assume that spatial data objects in ASPEN are point data. However, our design can be extended to support other spatial data types (e.g., lines, rectangles, and polygons).

3.1 Scatter Regions

To efficiently support spatial data objects in ASPEN, we transform the logical space used in CAN into a bounded physical space. Spatial objects are mapped onto this space according to their physical coordinates. Scatter regions are created in this physical plane as logical distribution units, each covering multiple zones. Their purpose is to preserve the spatial locality inherent in the data objects. Figure 4 illustrates a plane with 16 scatter regions. We define $\mathbb{S}$ to be the set of scatter regions in ASPEN (for symbol definitions see Table 1). Let $S_{addr} \in \mathbb{S}$ be a scatter region with address $addr$. This address is generated by interleaving the bit strings associated with the region's location. For instance, Figure 4 identifies a scatter region with address 1011, denoted as S_{1011}. Spatial data objects located in a specific scatter region are uniformly distributed across the zones within the scatter region (they cannot be distributed to adjacent regions). Data object B in Figure 4, for instance, can only be distributed to any zone within scatter region S_{1011}.

Notation	Definition
$\mathbb{S}$	The set of scatter regions in ASPEN.
S_{addr}	$S_{addr} \in \mathbb{S}$, with the scatter region address $addr$.
L_i	The load on the node i.
$LMAX_i$	Max load on a node i with stable scatter regions.
$LMIN_i$	Min load on a node i with stable scatter regions.
S_{init}	Initial scatter region size in ASPEN.

Table 1: Definition of notations in ASPEN.

The purpose of scatter regions is to simultaneously constrain the distribution of spatial data objects, while uniformly scattering them within the regions. Larger sized regions result in a more uniform distribution of spatial data objects. However, they also result in a larger search area for spatial range queries.

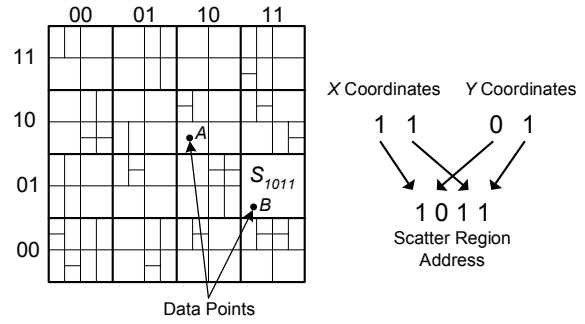


Figure 4: ASPEN plane with 16 scatter regions. Grids enclosed with thick lines are scatter regions while grids enclosed with thin lines are fractional zones.

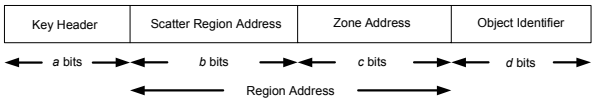


Figure 5: The data object key structure.

3.2 Spatial Data Object Key Generation Function

The *spatial data object key generation function* (or key generation function for short) is the basis of our methodology in that it enforces the constraints of scatter regions. First, we present the data object key structure that is used. Next, we illustrate the data object key generation procedure with an example.

The data object key generated by our function consists of a *scatter region address* for preserving spatial locality and a *zone address* to maintain good load balancing. Figure 5 illustrates the data object key structure, which is a bit string consisting of four parts: *key header*, *scatter region address*, *zone address*, and *object identifier* [15]. The length of each part is a bits, b bits, c bits and d bits, respectively. We term the scatter region address and the zone address ($b+c$ bits) collectively as the *region address*. The key header defines the user selected length of the scatter region address, reflecting an initial bound for the spatial data object distribution in ASPEN. The object identifier is concatenated to the key header and the region address to ensure the uniqueness of each data key. Note that the length of the region address also defines the system capacity (we can set the length of the region address to be very large to support a nearly infinite number of nodes in the system). Hence, the maximum number of zones in ASPEN is 2^{b+c} and the scatter region size is equal to $1/2^b$ of the complete plane.

To generate an object key for a spatial data object, the key generation function first maps the physical coordinates of the spatial data object to a scatter region in which the object is located, thus determining its scatter region address. Next, the zone address within the scatter region is decided by applying a consistent hash function such as SHA-1 [1] to the data object identifier. The final object key of the data object consists of the concatenation of the key header, the scatter region address, the zone address, and the object identifier.

Consider the following example to illustrate the key generation process for object A in the example plane of Figure 4. Assume that we build an ASPEN system with a region address length of 10 bits. The maximum number of zones is therefore $2^{10} = 1,024$. Consequently, the key header a must be $\lceil \log_2 10 \rceil = 4$ bits long to define the length of the scatter region address. The scatter region size in Figure 4 is set to one sixteenth $(1/16 = 1/2^4)$ of the full plane, thus the length of the scatter region address b is 4. The key header is then set to be "0100" in binary reflecting the length of the scatter region address. The length of zone address c is $10 - 4 = 6$ bits.

To obtain the data object key, the key generation function first sets the scatter region address to "1100" based on the physical location of A. Next the object identifier of A (e.g., spatial coordinates) is hashed. Assume the result is "101011" identifying the zone address. The final object key is the concatenation of the key header, the region address, and the object identifier of A: "0100 + 1100 + 101011 + object identifier".

The key generation function ensures the uniform distribution of spatial data objects within the scatter regions. More importantly, it provides a method to select the load distribution versus spatial locality preservation along a spectrum as a function of the length of the scatter region address in the key header.

3.3 Static Scatter Regions

In the following sections we describe the fundamental ASPEN system operations: node join and departure, spatial data object insertion and deletion, and the execution of spatial range queries. We start our description of these operations with the assumption that scatter regions are static (i.e., the value of the key header is constant at all nodes). Succeeding sections will extend these algorithms to support dynamic scatter regions which autonomously adjust the scatter region size in ASPEN.

3.3.1 Node Join and Departure

When a new node is about to join the system, an existing node (the bootstrap node) will be selected first. The bootstrap node randomly generates a destination point in the ASPEN plane for the new node. Next, the new node is routed to a node controlling the destination point. When the new node reaches this destination node, it checks the node and its immediate neighbors to find the best candidate node. To balance the number of data objects on each node, the node with the highest system load (defined in Section 3.4) will be selected to split into halves, each being controlled by one node. Finally, the new and the split node update their region addresses, and the routing information is updated among the new node, the split node and their neighbors.

When a node departs from ASPEN, the node explicitly hands over its state and associated data objects to one of its neighbor nodes. The neighbor node then merges with the departing node's zone using the original CAN algorithm. Note that the neighbor node being chosen cannot be in a different scatter region from the departing node.

3.3.2 Spatial Data Object Insertion and Deletion

CAN uses a unified hash function to distribute data objects. However, with the concept of scatter regions, spatial data object insertion and deletion require the use of our new key generation function to determine the scatter region in which the spatial data object should be located. Here we describe procedures for inserting and deleting spatial data objects in ASPEN.

A spatial data object to be inserted into the system must follow three steps: (i) ASPEN generates a data object key using the key generation function described in Section 3.2. (ii) The data object is routed to a zone in the destination scatter region with the region address closest to the region address of the data object key. (iii) When the data object reaches the destination zone, it is stored on the node controlling that zone.

The deletion of a spatial data object from the system requires

four steps: (i) ASPEN determines the scatter region in which the spatial data object is stored by applying the key generation function to the object identifier. (ii) ASPEN sends a data object deletion message to the destination scatter region. The message includes the object identifier of the data object. (iii) Because the spatial locality is not preserved within a scatter region, an exhaustive search is launched within the destination region, i.e., deletion messages are forwarded among zones within the same scatter region. An efficient, parallel message forwarding mechanism can be achieved through application-level multicast. We adopted the M-CAN algorithm [10], an efficient multicast algorithm that is able to scale to very large group sizes without requiring distribution trees or centralized control. (iv) If a data object is found on a node with the same object identifier, the node deletes the data object and a confirmation message is returned.

3.3.3 Spatial Range Queries

DHT systems provide a lookup function to perform exact match queries. The design of ASPEN focuses on providing similar search capabilities for spatial data. Specifically, we are interested in spatial range queries, which commonly occur in spatial data management applications. Algorithm 1 presents the spatial range query algorithm with static scatter regions. When a node in ASPEN receives a range query request, it first calculates how many scatter regions overlap with the query window (e.g., rectangle or polygon), see line 3. A unique query ID is assigned to the query on line 5. A query message is then submitted to each scatter region overlapping the query window. Query message routing from the source node to the scatter regions is accomplished via a greedy forwarding algorithm, as in CAN. After arriving at the scatter regions, the query messages are forwarded among the zones with the M-CAN algorithm (line 10). Lines 12 to 14 return any matching results to the querying node.

Algorithm 1 Spatial Range Query with Static Scatter Regions (N, q)

```
1: /* N is the querying node */
2: /* q is the query window (e.g., rectangle, polygon) */
3: invoke Find Overlapping Scatter Regions(q)
4: /* Finding the set of scatter regions S' ⊂ S, AND S' over-
   lapped by the query window */
5: invoke Assign Query ID(q)
6: /* Assign a unique ID qid to the query */
7: for each scatter region S ∈ S' do
8:    invoke Route Query(N, S, qid, q)
9:    /* Forward the query message to a node N' within the scatter
      region */
10:   invoke M-CAN (N', S, qid, q)
11:   /* Perform M-CAN within the scatter region */
12:   for each data object o matches q do
13:      return o to N
14:   end for
15: end for
```

3.4 Dynamic Scatter Regions

The algorithms presented so far assume a static scatter region layout. However, one of the characteristics of P2P systems is that they are highly dynamic, self-organizing collections of computers. Static scatter regions are likely to be sub-optimal in providing load balance over the long run. We therefore extend our methodology to provide dynamic adaptation as follows. Let the load L_i of a node i denote the resource usage on that node at that time [6](Table 1).

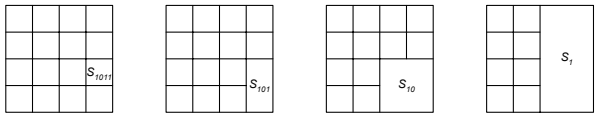


Figure 6: Example of 3 scatter region merges starting with S_{1011} in succession given 16 initial scatter regions.

This might represent, for instance, used disk space, or other user identified resources. The load L_i on a node i exceeding an upper threshold represents a performance bottleneck. Since data objects within the same scatter region are uniformly distributed, this condition also implies that a system overload in all the zones of the scatter region is very likely to occur. However, neighboring scatter regions may be less loaded and we can improve load balancing by merging adjacent scatter regions. Data objects are then re-hashed into the newly merged scatter region. Based on this observation, we propose our dynamic scatter region design, which adjusts the size of scatter regions adaptively to achieve better load balancing.

We propose two thresholds, $LMAX_i$ and $LMIN_i$, to represent the maximum and minimum load managed by a node i. We define the load within the range of $[LMIN_i, LMAX_i]$ to be the *stable load range* on node i. Let S_{init} be the initial scatter region size (i.e., the initial value of the key header) in ASPEN. Note that all nodes in ASPEN use the same value of S_{init} to ensure system consistency during scatter region operations. S_{init} also defines the minimum scatter region size allowed in ASPEN and no further scatter region split operations can be performed. The values of the thresholds $LMAX_i$, $LMIN_i$ and S_{init} are user selectable.

We define a specific order in which scatter regions are merged in ASPEN such that the operation can be executed in a distributed manner with minimal communications overhead. Figure 6 shows three successively performed merge operations starting with scatter region S_{1011} from 16 initial scatter regions. Split operations follow the same order in reverse. The ordering allows the recording of a change in the scatter region size on nodes and data objects by simply modifying the value of the key header (a decrease by 1 for a merge operation and an increase by 1 for a split operation). More significantly, this global ordering ensures system consistency in a dynamic, distributed environment since all nodes conform to the same order when calculating the scatter region size and location in scatter region operations.

If there is a node i with $L_i > LMAX_i$, then the node load is above capacity. Consequently, the node initiates a scatter region merge operation with its neighboring regions. In addition, the data objects within the newly merged scatter region will be uniformly re-distributed to improve the load balance. A drawback of such a region merge is that more messages must be generated in order to forward spatial range queries in a larger scatter region. Consequently, if the value of L_i on a node i falls below $LMIN_i$, the node launches a scatter region split operation, with the reverse semantics of the merge operation, to improve the spatial range query efficiency.

Next we detail the scatter region merge and split operations. We also present the modified system operations (e.g., join and departure) and the spatial range query algorithm for the dynamic scatter region design. Finally, we discuss the spatial range query complexity and the system stability.

3.4.1 Scatter Region Merge and Split

DHT systems have no central control mechanism, and therefore nodes may receive split and merge messages in a different order

from the actual sequence in which the messages were generated. However, if scatter region operations are executed out-of-order, the system state may become inconsistent, i.e., the scatter region size may be recorded differently at various nodes. To ensure consistency, ASPEN timestamps each scatter region operation message. Every node also remembers the time stamp from the latest scatter region operation it performed. When receiving a new message, a node will check the time stamp against its historical information. Only messages having newer time stamps will be accepted and executed on the node; other messages will be discarded.

If a node i experiences a load $L_i > LMAX_i$, the node launches the scatter region merge operation shown in Algorithm 2. The node submits merge messages that include a time stamp and the new key header to nodes located in the merged scatter region via M-CAN multicast. When a node receives a scatter region merge message, it checks the time stamp first. If system consistency is assured, the node launches the scatter region update operation illustrated in Algorithm 3. First, the local node information is updated with the new key header and the time stamp from the message. Second, the node updates the key header of each data object it stores. Next, the zone address of all data objects are re-hashed and new object identifiers generated with the new scatter region setting. Finally, the node sends data objects to new zones according to their new region addresses.

Algorithm 2 Scatter Region Merge

```
1: for any node i do
2:   if L_i > LMAX_i then
3:     invoke Calculate Scatter Region Address
4:     for nodes in the new merged scatter region do
5:       invoke Scatter Region Update(ts,header,"Merge")
6:       /* ts is the time stamp of the scatter region operation
         message */
7:       /* header is the new key header */
8:     end for
9:   end if
10: end for
```

Algorithm 3 Scatter Region Update (ts, $header$, $flag$)

```
1: /* ts is the time stamp of scatter region operation message */
2: /* header is the new key header */
3: /* flag is to indicate whether a scatter regions merge or split
   message */
4: if invoke Check Time Stamp(ts) = true then
5:   /* Check if it is the most recent scatter region operation mes-
     sage received */
6:   invoke Update Key Header(keyheader)
7:   invoke Update Time Stamp(ts)
8:   for each data object o stored on the node do
9:     invoke Update Data Object(keyheader)
10:    /* Update the key header the data object, and re-hash to
       get the new zone address of the data object */
11:    invoke route(o)
12:    /* Route the data object according to its new region ad-
       dress */
13:  end for
14: end if
```

If the load L_i of a node i falls below $LMIN_i$ while the size of the scatter region in which the node is located is greater than S_{init}, the node launches the scatter region split operation shown in Algorithm 4. The current scatter region will be evenly split into halves by observing the reverse ordering of previous merge operations. The node first forwards scatter region split messages with a time stamp and the new key header to nodes in the same scatter region using the M-CAN multicast protocol. When a node receives the split message, it checks for system consistency. If it is assured, the node launches the scatter region update operation as described in Algorithm 3.

Algorithm 4 Scatter Region Split

```
1: for any node i do
2:   if L_i falls below LMIN_i AND i.scatterregionsize >
     S_init then
3:     invoke Calculate Scatter Region Address
4:     for nodes in the new split scatter region do
5:       invoke Scatter Region Update(ts, Addr_sc,"Split")
6:       /* ts is the time stamp of the scatter region operation
         message */
7:       /* Addr_sc is the new scatter region address */
8:     end for
9:   end if
10: end for
```

3.4.2 Node Join and Departure

The node join algorithm with static scatter regions is adapted to the dynamic case as follows. The bootstrap node now generates a destination region address for the new node based on S_{init}. It also records an absolute time stamp (e.g, 0) as the initial time stamp of the scatter region operation on the new node. Next, when the new node reaches the destination zone, the node controlling the destination zone checks for key header consistency with the new node. If key header values are different from each other, it implies that either merge or split operations have taken place. The node updates its key header and the time stamp to reflect the latest scatter region operation. Other parts of the node join algorithm are the same with the static scatter region design. The algorithm of the node departure with dynamic scatter regions does not require any modification from the static design.

3.4.3 Spatial Data Object Insertion and Deletion

Spatial data object insertion with dynamic scatter regions can be supported with a few modifications to the static scatter region design. First, ASPEN assigns a data object key to the data object by using our key generation function based on S_{init}. Next, when the spatial data object reaches the destination zone, the node controlling the destination zone checks for key header consistency with the spatial data object. If the key header values are different from each other, it implies the scatter regions have changed. In such a case the node updates the key header of the data object. The node also re-hashes the object identifier to obtain a new zone address for the data object. After the re-hashing the data object will be routed according to its new region address.

To accommodate spatial data object deletion with dynamic scatter regions, we made the following modifications to the static scatter region design. First, S_{init} is used when ASPEN determinates the scatter region in which the spatial data object is located. The deletion message also records the value of S_{init}. Second, every node in the scatter region forwarding the deletion message will check its scatter region setting with the deletion message. If the scatter region in which the node is located is larger than the value of S_{init} in the deletion message, the node updates S_{init} in the deletion message to enlarge the area being forwarded by M-CAN, and keeps forwarding the message in the updated area.

3.4.4 Spatial Range Queries

Algorithm 5 presents the spatial range query algorithm with dynamic scatter regions. When a node in ASPEN receives a spatial range query request, the node first calculates how many scatter regions overlap with the query window based on S_{init} as shown in line 3. Lines 5 to 8 assign a unique query ID and record S_{init} with the query. A query message is then submitted to each scatter region overlapping the query window. Every node forwarding the query message will check its scatter region setting against the value of S_{init} in the query message (line 13). If the scatter region in which the node is located is larger than the value of S_{init} in the query message, the node updates S_{init} in the query message (line 14) to enlarge the area covered by M-CAN, and keeps forwarding the message. Any matching results will be returned to the querying node.

Algorithm 5 Spatial Range Query with Dynamic Scatter Regions (N, q)

```
1: /* N is the querying node */
2: /* q is the query window (e.g., rectangle, polygon) */
3: invoke Find Overlapping Scatter Regions(q)
4: /* Finding the set of scatter regions 𝕊′ ⊂ 𝕊, AND 𝕊′ overlapped by the query window, based on S_init */
5: invoke Assign Query ID(q)
6: /* Assign a unique ID qid to the query */
7: invoke Assign Key Header(q, S_init)
8: /* Assign S_init to the query recording the initial scatter region setting */
9: for each scatter region S ∈ 𝕊′ do
10:    invoke Route Query(N, S, qid, q)
11:    /* Forward the query message to a node within the scatter region */
12:    for each node N′ in the scatter region do
13:       if N′.scatterregionsize > q.S_init then
14:          invoke Update Scatter Region(q)
15:       end if
16:       for each data object o matches q do
17:          return o to N
18:       end for
19:       invoke M-CAN (N′, S, qid, q)
20:       /* Perform M-CAN within the scatter region */
21:    end for
22: end for
```

3.5 Query Complexity Analysis

Spatial range query execution in ASPEN consists of two phases: the first phase represents sending query messages from the querying node to the edge of scatter regions overlapping the query window. The second phase is the M-CAN message forwarding within the scatter regions. The design of ASPEN focuses on the efficiency and user experienced query latency of spatial range queries. We utilize two metrics, the hop count and the message count, to measure the query complexity. The hop count for a spatial range query denotes the number nodes on the longest path along the query message routing and M-CAN forwarding of a spatial range query, which indicates the query latency experienced by a user. The message count represents all the network traffic generated by a query. Below we present the query complexity analysis in terms of the hop count and the message count.

Given a d dimensional space partitioned into n equal zones, individual nodes maintain $2d$ neighbors, and the average routing path length is $(d/4)(n^{1/d})$ [10]. The first phase of the routing can be achieved in O($n^{1/d}$). For the second phase of the query, let us assume that zones are equally partitioned in scatter regions. Let there be e scatter regions in the system, each containing n/e zones. In a d dimensional space, the hop count is O$(n/e)^{1/d} \approx$ O$(n^{1/d})$. Hence the total complexity of the hop count for an equally partitioned space is O($n^{1/d}$) + O($n^{1/d}$) = O($n^{1/d}$), which indicates that ASPEN retains the same excellent scalability property for search operations as the original CAN.

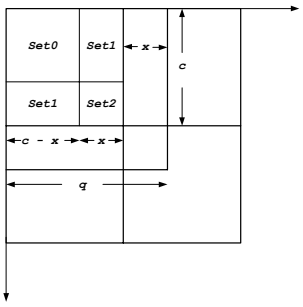


Figure 7: Message Count Analysis in ASPEN

To analyze the message count in query processing, let us assume that zones are evenly distributed in the ASPEN space. Let c denote the side of each zone. Let q denote the length of each side of a spatial range query Q where $q > c$. Then q can be represented as $i \times c + x$, where $x \subset [0, c)$ and i is an integer. Without loss of generality let us consider the case where the top-left corner of query q is located somewhere within the top-left zone of ASPEN as shown in Fig 7 [8]. It can be verified that if the top-left corner of Q is inside Set0 then Q covers $(i+1)^2$ zones, for Set1 Q covers $(i+1)^2+i+1$ zones, and for Set2 it is $(i+2)^2$. Assuming uniform distribution of spatial range queries in ASPEN, on the average Q covers $(q+c)^2/c^2$ zones.

We can assume $q^2/c^2 = n$ where n denotes the total number of nodes in ASPEN (i.e., Query Q covers the complete space of ASPEN) as the worse case of spatial range query, hence the number of messages in the second phase of the query is O$((q+c)^2/c^2) \approx$ O(n). Since the message count in the first phase of the query is O($n^{1/d}$) in a d dimensional space, thus the total complexity of the message count is O(n).

4. EXPERIMENTAL EVALUATION

To evaluate the performance of ASPEN we performed extensive simulations. The metrics of interest were the load balance and spatial range query performance as functions of the scatter region size. We performed our experiments on both synthetic and real world data sets. The results indicate that our design of scatter regions achieves good load balancing with a wide variety of scatter region settings. Our ASPEN approach also exhibits excellent scalability of the spatial range query performance. In Section 4.1 we start by describing both the synthetic and real world data sets used in our simulation. We detail our simulator implementation in Section 4.2. Experimental results and an analysis are presented in Section 4.3.

4.1 Experimental Data Sets

We obtained a real geotechnical data set with highly skewed spatial distribution to verify the performance of ASPEN. To test the system scalability with a large number of nodes and data objects, we also generated a synthetic data set with up to 10^5 spatial data objects in our experiments. We describe the characteristics of both data sets in succeeding sections.

4.1.1 Kobe Data Set

Our primary data set was provided by Kobe University, Japan[1]. The data set represents geotechnical point data and contains information about slightly more than 4,000 borehole records with coordinate values located in Kobe County. Foremost examples of geotechnical information, geotechnical boreholes are vertical holes drilled in the ground for the purpose of obtaining samples of soil and rock materials and determining the stratigraphy, groundwater conditions and/or engineering soil properties. We will refer to these records as the Kobe data set.

4.1.2 Synthetic Data Set

The Kobe data set is a highly skewed, with an increased density in the populous, coastal areas. Additionally, the number of data objects is relatively small (approximately 4,000). To test our design with a larger number of data objects, especially to capture the query performance of ASPEN, we created a synthetic data set consisting of 10^5 point objects. The data was organized into ten clusters, each centered around a randomly selected, geographical centroid. The points associated with each cluster were generated based on a normal distribution of the distance from the corresponding center point location. Thus, data points were denser around the center point and became sparser with increasing distance from the center point. The standard deviation for this synthetic data set was five.

4.1.3 Query Window Data Set

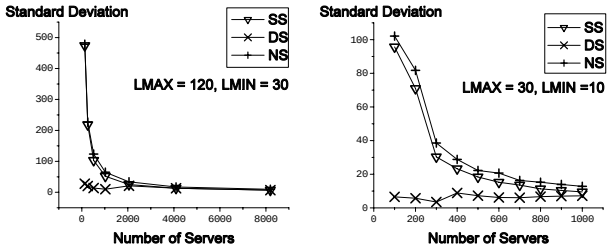
We generated a set of 1,000 rectangular query windows to verify the spatial range query performance of our design. Each query window dimension and location was chosen as follows. We first computed a query window size based on a normal distribution. Then we randomly selected a point (x_1, y_1) as one rectangle corner vertex, and a value x_2 inside the global boundary as the x-value of the corresponding diagonal vertex. Based on x_1, x_2, y_1 and the window size, we calculated y_2, which finalized the query window location. Our window data set consisted of 1,000 queries, with an average query window size of 5% of the plane size, based on a normal distribution with a standard deviation of two.

4.2 Simulator Implementation

For simplicity, we assume that the data objects stored in ASPEN all have the same weight (i.e., utilize the same amount of resources). The number data objects stored on a node i is used to represent the load L_i and the values of $LMAX$ and $LMIN$ are the same on all nodes. The ASPEN simulator implements the core functionality of ASPEN: the scatter region management, the key generation function and the spatial range query algorithm for both static and dynamic scatter regions [2]. We have performed two main categories of experiments: (a) measuring the system load and (b) determining the spatial range query performance. The ASPEN simulator is structured into three main components: the *2-D plane generator*, the *spatial data loader*, and the *performance monitor*. The 2-D plane generator implements our node insertion algorithms, creating a 2-dimensional ASPEN plane fully populated with a given number of nodes. The spatial data loader computes the keys of the spatial input data objects using the key generation function of Section 3.2. It also assigns data objects to nodes according to the spatial data object insertion algorithm. Finally, the performance monitor retrieves the number of data objects on each node to measure the load balance in ASPEN. To perform spatial range query experiments, the performance monitor executes the query window data set against the loaded spatial objects. For each query window, a node is randomly selected as the starting point to launch the search. The performance monitor executes our spatial range query algorithms and reports the number of messages generated and the longest path (in hops) as the application-level query routing cost and response time, respectively.

[1] http://www.kobe-u.ac.jp/

[2] We appreciate Dr. Sylvia Ratnasamy for providing us with the CAN simulation code.



(a) Synthetic Data Set (b) Kobe Data Set

Figure 8: The system load as a function of number of nodes in the system.

4.3 Simulation Results

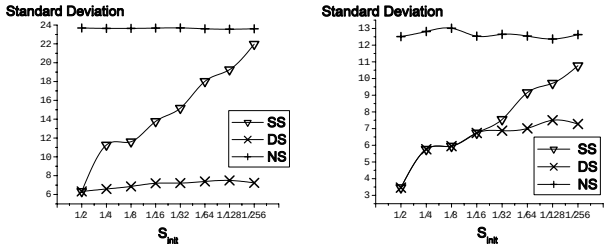
We constructed an ASPEN system with a maximum capacity of 2^{16} nodes. The experiments measured two metrics: the load balance and the spatial range query performance. The simulation was executed on a workstation with 1 GB memory and a 1.8 GHz AMD Opteron processor. In the discussion of the results, we use SS to denote static scatter regions, DS to denote dynamic scatter regions, and NS to denote a baseline setup without scatter regions, where data objects are directly mapped onto nodes without using our key generation function. For each experimental configuration, the ASPEN simulator executes 20 times and reports the average results.

4.3.1 Load Balance

We use the standard deviation of the load L on all nodes to represent the metric of the load balance. A smaller value indicates better load balancing because data objects are more uniformly distributed among nodes.

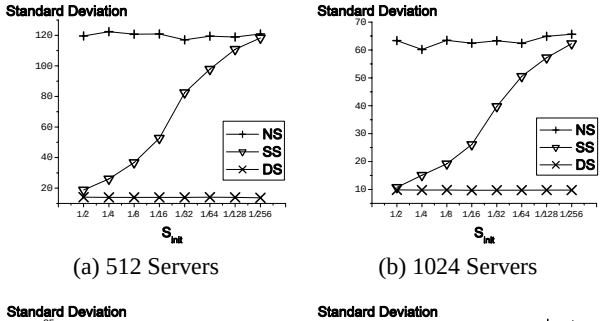
We were first interested in how the number of nodes affects the load balance. Figure 8 plots the system load as a function of the number of nodes with both the synthetic and the Kobe data sets. We initialized S_{init} to be 1/64 of the whole plane. The results clearly show that a higher number of nodes results in an improved load balance (i.e., more uniform load) with both the synthetic and the Kobe data sets. Static scatter regions (SS) perform slightly better than no scatter regions with the Kobe data set. However, dynamic scatter regions (DS) show a significant improvement across the full range of servers, exhibiting consistently the best and most stable load balance.

Next, we explored the relationship between the load balance and the initial scatter region size S_{init}. We varied the value of S_{init} from 1/2 to 1/256 of the complete plane in divisions of 2. Figure 9 plots the experimental results with the Kobe data set. The number of nodes in the system is 512 and 1024, respectively. We set $LMAX$ to be 30, and $LMIN$ to be 10. Correspondingly, Figure 10 plots experimental results with the synthetic data set. Here the number of nodes in the system is 512, 1024, 2048, and 4096, respectively. We set $LMAX$ to be 120, and $LMIN$ to be 30. Figures 9 and 10 illustrate that the system achieves excellent load balancing with large initial scatter region sizes and the load balance deteriorates as the initial scatter region size decreases. Intuitively

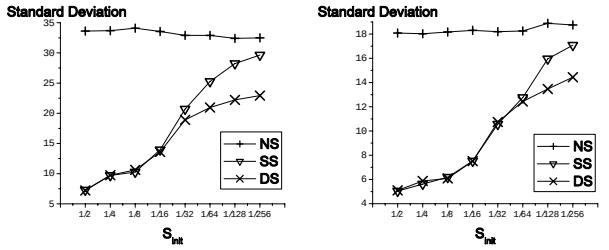


(a) 512 Servers (b) 1024 Servers

Figure 9: The system load as a function of S_{init} with the Kobe data set.



(a) 512 Servers (b) 1024 Servers



(c) 2048 Servers (d) 4096 Servers

Figure 10: The system load as a function of S_{init} with the synthetic data set.

with a smaller initial scatter region size, the space for uniformly distributing data objects becomes more constrained with static scatter regions, which adversely affects the load balance. On the other hand, as shown in Figures 9(b), 10(c) and 10(d), the load balance stabilizes once the scatter region size is reduced below a certain value. This illustrates the effect of scatter region merges (or splits) once the value of L deviates from the stable load range. Data objects are re-distributed and the value of L is constrained in the stable range. Hence the load balance is excellently maintained with dynamic scatter regions. The results also demonstrate that for all possible initial scatter region sizes, both static and dynamic scatter regions maintain better load balancing than the cases without scatter regions.

Figure 11 demonstrates snapshots of the scatter region distribution and the number of objects on each node with the Kobe data set. The number of nodes in the system is 512. We set $LMAX$ to be 30, $LMIN$ to be 10, and S_{init} to be 1/32. Figure 11(b) shows the initial data distribution with S_{init}. Some of the regions are heavily loaded while others are empty. Figure 11(c) shows the benefits of dynamic scatter regions: the number of data objects on all nodes is constrained to the stable range through merge operations and hence the load balance is much improved.

4.3.2 Query Performance

We use the hop and message counts to represent the metric of

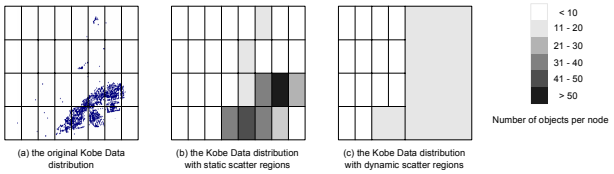
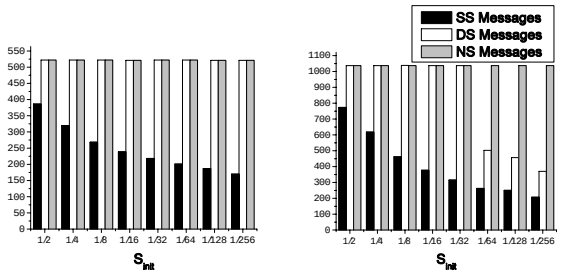
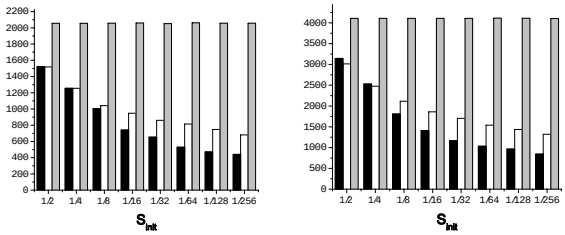


Figure 11: Snapshots of the scatter region distribution in ASPEN with the Kobe data set



(a) 512 Servers (b) 1024 Servers

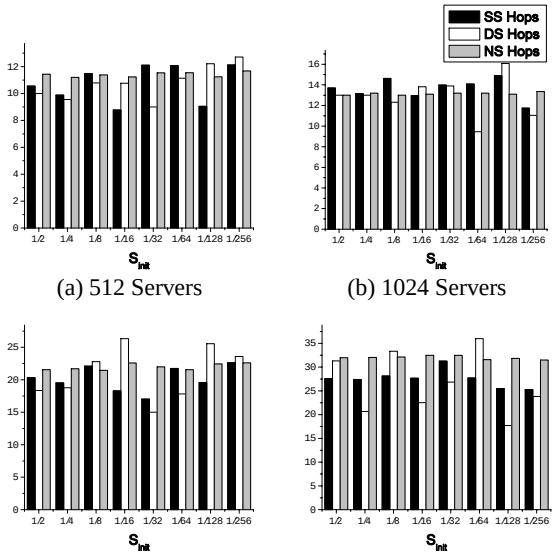


(c) 2048 Servers (d) 4096 Servers

Figure 12: The query message count as a function of S_{init} with the synthetic data set.

spatial range query performance. Our simulation reports the hop and message counts as described in Section 3.5. A smaller number for the hop and message counts represents a better spatial range query performance. We measured the query performance of ASPEN with the synthetic data set. Figures 12 and 13 illustrate the average hop and message counts for the ASPEN system.

We make several observations from Figure 12. As a general trend, the number of query messages drops in correspondence with the decrease of the initial scatter region size. Our explanation for this difference is that a larger scatter region implies a higher number of nodes per region. Since spatial range queries must be forwarded within scatter regions overlapping the query window, more messages are required in order to forward queries in a larger scatter region. Figure 12(a) shows an exception to this general trend. Here, a considerable number of query messages is required in a wide variety of dynamic scatter regions with 512 servers. This is because with 10^5 data objects distributed over 512 servers, our dynamic scatter region design continues to merge scatter regions until the whole space is a single region in order to constrain the value of L on all nodes to be within the stable range. Therefore, an exhaustive search is required for spatial range queries. This can be avoided by increasing the value of $LMAX$ so that the system has a wider stable range. Second, the query message count with dynamic scatter regions is higher than the one with static scatter regions across a wide range of initial scatter region sizes. This is because dynamic scatter regions will perform a scatter region merge to achieve better load balancing when there is a node i with $L_i > LMAX_i$. This results in a higher number of messages for spatial range queries.



(a) 512 Servers (b) 1024 Servers

(c) 2048 Servers (d) 4096 Servers

Figure 13: The query hop count as a function of S_{init} with the synthetic data set.

On the other hand, Figure 13 shows a steady query hop count with various scatter region sizes. This is expected from our spatial range query complexity analysis in Section 3.5. Our simulation works on a 2-D space, which shows the hop count is only a function of the number of nodes in ASPEN. Given the parallel processing capabilities of P2P systems, the ASPEN system provides advantages in processing of large numbers of concurrent queries while maintaining good response times.

Our experimental results show that a smaller scatter region size reduces the number of messages required with spatial range queries. This is, however, causing a worse load balance. On the other hand, a large scatter region size is able to distribute spatial data objects more uniformly but is requiring more messages for spatial range queries. Hence the relation of load balance versus spatial range query performance can be selected on a spectrum. Users are able to choose preferred values of S_{init}, $LMAX_i$, and $LMIN_i$ that reflect which feature is more important in their application. Specifically, our design of dynamic scatter regions is able to adaptively distribute the system load. Hence a small initial scatter region size, which performs well for spatial range queries, can be adaptively adjusted to achieve a specific level of load balancing.

5. CONCLUSIONS AND FUTURE DIRECTIONS

DHT systems have generated intense interest in research because their robustness, load balancing, and scalability are desirable for large-scale systems. We have presented ASPEN, a novel spatial DHT-based system, that preserves both spatial locality information and load balancing in distributed environments. Specifically, we have integrated our design with CAN by introducing the concept of scatter regions to efficiently support spatial data. We have presented ASPEN system operations, such as spatial data object insertion and deletion, with both static and dynamic scatter region settings. We also have illustrated the practicality of supporting spatial range queries with constrained load balancing and the results that we observed are very promising.

We plan to extend our work in several directions. First, we will integrate dynamic network-level topology models to acquire more accurate information about the trade-off in performance when adjusting parameters along the spectrum of spatial locality preservation versus data load balancing. Second, we intend to study the query success rate under dynamic network environments. Furthermore, the support of more types of spatial queries, such as the nearest neighbor queries, is worth exploring.

6. REFERENCES

[1] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 1995.

[2] J. Aspnes, J. Kirsch, and A. Krishnamurthy. Load Balancing and Locality in Range-Queriable Data Structures. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, 2004.

[3] T. Bially. A Class of Dimension Changing Mapping and Its Application to Bandwidth Compression. In *Ph.D. Dissertation Polytechnic Inst. of Brooklyn*, June 1967.

[4] R. Finkel and J. Bentley. Quadtree: A data structure for retrieval on composite keys. *ACTA Informatica*, 1974.

[5] J. Gao and P. Steenkiste. An Adaptive Protocol for Efficient Support of Range Queries in DHT-Based Systems. In *12th IEEE International Conference on Network Protocols*, 2004.

[6] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Dynamic Structured P2P Systems. In *Proceedings of the 23rd Conference of the IEEE Communications Society*, 2004.

[7] J. Hellerstein, S. Ratnasamy, and S. Shenker. Range Query over DHTs. Technical Report IRB-TR-03-009, Intel Research, Berkeley, June 2003.

[8] D. V. Kalashnikov, S. Prabhakar, S. E. Hambrusch, and W. G. Aref. Efficient evaluation of continuous range queries on moving objects. In *13th International Database and Expert Systems Applications Conference*, 2002.

[9] A. Mondal, Y. Lifu, and M. Kitsuregawa. P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments. In *EDBT Workshops*, pages 516–525, 2004.

[10] S. Ratnasamy. A Scalable Content-Addressable Network. In *Ph.D. Dissertation University of California Berkeley*, 2002.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[12] C. Schmidt and M. Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *12th Symposium on High-Performance Distributed Computing*, 2003.

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of SIGCOMM*, 2001.

[14] E. Tanin, A. Harwood, H. Samet, S. Nutanong, and M. T. Truong. A Serverless 3D World. In *Proceedings of the 12th International Symposium of ACM GIS*, 2004.

[15] S. Zhou, G. R. Ganger, and P. Steenkiste. Location-based Node IDs: Enabling Explicit Locality in DHTs. Technical Report CMU-CS-03-171, Carnegie Mellon University, 2003.

[16] R. Zimmermann, W.-S. Ku, and H. Wang. Spatial data query support in peer-to-peer systems. In *28th International Computer Software and Applications Conference*, 2004.