

Spatial Data Query Support in Peer-to-Peer Systems*

Roger Zimmermann, Wei-Shinn Ku, and Haojun Wang
Computer Science Department
University of Southern California
Los Angeles, California 90089, United States
{rzimmerm, wku, haojunwa}@usc.edu

Abstract

Recently distributed hash table (DHT) mechanisms have been proposed to manage data in very large, structured peer-to-peer (P2P) systems. DHT algorithms provide efficient exact-match object search capabilities without requiring global indexing and are hence extremely scalable. However, range queries – which are very common with spatial data – cannot be executed efficiently in these systems because the adoption of uniform hash functions to ensure excellent load balancing unfortunately destroys any existing data locality. In this report we propose a novel technique to preserve spatial locality information while also keeping some of the load balancing properties of DHT based systems. We describe our design as an extension of content-addressable networks (CAN) and illustrate the feasibility of supporting spatial range queries.

1 Introduction

The recent research work in the area of structured peer-to-peer (P2P) systems has produced novel approaches for the distribution of large data sets in a decentralized manner. In systems such as CAN [4], Chord [6], and Pastry [5], node identifiers are usually randomly assigned as nodes join. Distributed hash tables (DHTs) allocate data objects to the nodes in the system with no central control. One of the goals is to distribute data objects near uniformly in the system, resulting in superb scalability, load balance and robustness. Each node only needs to maintain a small routing table with the contact information of a few neighboring nodes. The process of locating a particular data object is very efficient and in an n -node system on average requires only $O(\log n)$ search steps.

Some of the large scale data sets now available are geographic in nature. Applications such as Geographic Information Systems utilize these spatial data, integrating them

with textual and other information and creating innovative and powerful end user systems. By their very nature, inquiries about spatial data commonly concern a certain range or area, for example queries about intersections, containment, and nearest neighbors. Therefore, unlike with exact match queries, the spatial relationship of data objects, and therefore the *spatial locality* information needs to be preserved when data objects are distributed. However, the DHT mechanisms adopted by structured P2P systems tend to distribute data objects uniformly within the domain and destroy the spatial locality information. Consequently, spatial data queries cannot be supported efficiently. On the other hand, if we select a non-uniform hash function, some areas of the space might contain many data objects and while others will manage almost none. An area containing a large number of data objects usually results in many queries being performed in that area, therefore this may result in a performance bottleneck. Consequently, 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 approach that constructs the key for each spatial data object with three components: a fixed part based on the location of the data object, a random part that scatters the data object within a particular region of the space, and an object identifier based on the data content. We provide the initial design of a decentralized system to store spatial data objects with the preservation of spatial locality information and constrained load balance. We also describe how spatial range queries are supported in this structured P2P environment.

The rest of this paper is organized as follows. The related work is described in Section 2. We introduce our hash function and how it is used to generate data object keys in support of spatial data queries over DHTs in Section 3. We discuss the conclusions and future work in Section 4.

2 Related Work

Recently, Harwood et al. [3] proposed a design for efficiently querying spatial data over structured P2P systems. Their work is closely related to our approach. Based on the

*This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), CMS-0219463, and unrestricted cash/equipment gifts from Intel, Hewlett-Packard and the Lord Foundation.

Quadtree algorithm [2], their design recursively divides a two dimensional (2-D) space into smaller grids. Each grid space includes a control point that is responsible for the data objects within its grid area. The 2-D space is thus transformed into a tree structure that stores spatial data objects. Each control point is then hashed to a node in the underlying Chord ring [6]. Although this design can preserve spatial locality, we argue that – by directly mapping control points to nodes in Chord – the system cannot achieve good load balance: if a grid area contains a lot of data objects, a single control point will be responsible for all of them. Since every level of the tree structure can store data objects, performing a spatial range query requires the system to search an entire tree branch from the root node down to the leaves to find objects that intersect with the query window. Furthermore, every query needs to be propagated to the top-level nodes, which also implies a performance bottleneck at the higher level nodes in this design.

3 Spatial Range Query Design

Our design for supporting spatial data queries over structured P2P systems is inspired by work on scalable content-addressable networks (CAN) [4]. The focus is on the key assignment scheme and the hash function for storing data objects in the system. We also address the procedures for node and data object manipulation.

3.1 Mapping a Physical Space to a CAN Space

CAN introduced a novel approach to creating a scalable indexing mechanism in a P2P environment. It creates a logical d -dimensional Cartesian coordinate space divided into zones, where zones can be dynamically partitioned or merged as nodes join and leave. Each zone in this space is addressed with a Virtual Identifier (VID), which is calculated from the location of a zone in the logical space. Each node in the system is responsible for storing all data objects assigned to a specific zone. Figure 1 shows an example 2-D space partitioned into 7 CAN zones.

CAN nodes operate without global knowledge of the space and communicate through messages. A CAN node maintains a routing table that consists of the IP addresses and logical zone areas of its immediate neighbors. For each CAN message with the destination coordinate generated by a particular node in the system, the node will consult its routing table and simply apply a greedy forwarding algorithm to route the message to the destination zone.

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 another node (also called the bootstrap node), which is already a member of the CAN system. Second, the new node randomly chooses a point in the logical CAN

space, and the bootstrap node routes the new member to the destination zone which covers that point. Third, the destination zone is split into two zones, each being controlled by one node. Finally, the neighbors of the split zone will be notified to update their neighborhood routing information.

When a data object is to be inserted into a CAN system, the application generates a key based on the object and inserts it as a $\langle \text{key}, \text{value} \rangle$ pair. The key is then mapped onto a point P in the CAN space by using a uniform hash function. The key-value pair is stored at the node which owns the zone within which the point P is located. For the retrieval of a value, the same hash function is applied to the key in order to regenerate the point P in the logical CAN space. The request is then routed to the zone which owns the point P , to retrieve the value.

The original design of CAN constructs a pure logical coordinate space and adopts a uniform hash function for routing and indexing data objects in a distributed environment. Our work extends this virtual space with physical spatial meaning. The design introduces a new hash function for mapping spatial data objects onto nodes over a modified CAN system to allow efficient spatial data query execution while at the same time considering load balance.

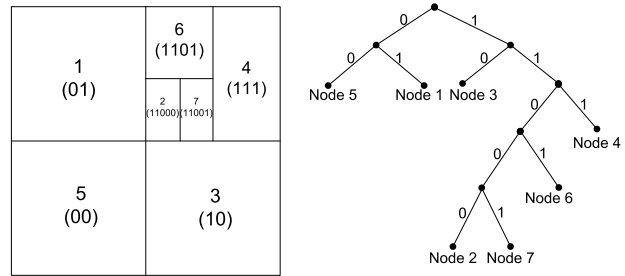


Figure 1. A CAN plane and its VID tree.

3.2 Data Object Hashing

To preserve the spatial locality of data objects, a hash function needs to assign keys of objects with close proximity within related branches of the VID tree. Conversely, the load balance of the system can be improved by distributing adjacent data objects within a larger area. To this end, the keys generated by our hash function consist of a fixed component for preserving spatial locality and a randomized component to control limited load balancing. The sizes of the two components are user selectable and together determine the *maximum number of zones*, i.e., the total node capacity. The fixed component functions as an address to the scatter regions (and hence determines the *scatter region size*) while the randomized part addresses the zones within a scatter region where object keys are distributed (see Fig. 2). A virtual identifier tree (VID tree) is created with its height

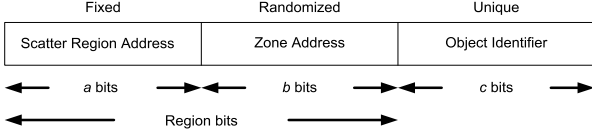


Figure 2. The data object key structure.

determined by the scatter region size (illustrated in Figs. 3 and 4). Objects can enter the system at any node, which generates data keys based on the physical coordinates of the objects and the VID tree information. Collaboratively, the nodes route the objects to their destination zones for storage. Specifically, the key for each data object is a bit string consisting of three parts: scatter region address, zone address, and object identifier (Fig. 2) [7]. The scatter region bit string is determined by the physical coordinates of the input data object and mapped to a leaf node value in the VID tree. The value of zone bit string is decided randomly and the object identifier is the data content hash result of a consistent hash function such as SHA-1 [1]. The length of each component is a bits, b bits, and c bits, respectively. Hence, the maximum number of zones is $2^{(a+b)}$ and the scatter region size is equal to $\frac{1}{2^a}$ of the complete space. Consider the following example to illustrate the hashing process. Assume that the maximum number of zones is 1,024. Hence, the first two parts of the key bit string are ten bits long and we call them collectively the *region bits* (see Fig. 2). If the scatter area size is defined as one thirty-second ($\frac{1}{32} = \frac{1}{2^5}$) of the complete space (i.e., the zone division shown in Fig. 4), then the first five bits must be set for a deterministic scatter zone. Provided the physical location (e.g., center point) of a data object is mapped to zone A in Figure 4, the hash function sets the first five bits of the key to “11000” (the VID tree label for zone A) and randomly decides the other five region bits. The content hash result is concatenated to the region bits to ensure the uniqueness of each data key. Consequently, assuming that the content hash identifier is also five bits, the generated key might be “11000 01011 01101” and it will be routed to a deterministic zone. If the specified zone, “11000 01011,” does not exist, the key will be routed to the zone with the closest key value. In this example, a scatter region can accommodate at most 32 zones.

Users are able to determine the scatter region size by selecting the number of scatter region bits. Therefore, the relationship between data locality and load balance can be determined along a spectrum. If we reduce the number of scatter region bits, the size of the scatter region will be enlarged and the load balance of the system will be improved. However, the search area for a range query will also increase and hence the search performance will degrade. We explain this trade-off in more details in the next section.

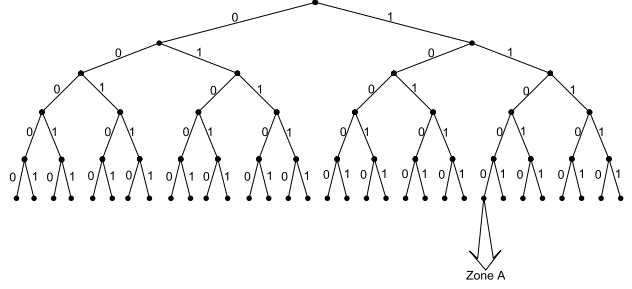


Figure 3. A full VID tree with the scatter zone size to be 1/32 of the complete space.

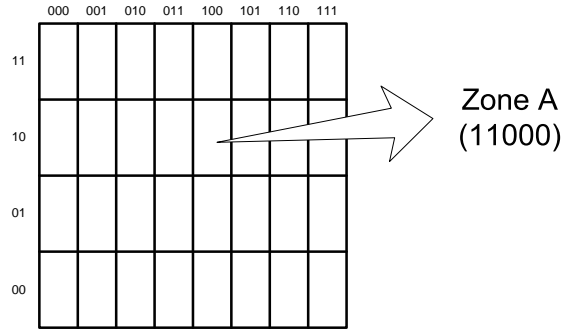


Figure 4. Mapping of a full VID tree to the CAN plane. In this example, each zone covers 1/32 of the space and its zone VID can be generated by interleaving the two bit strings associated with its zone.

3.3 System Operation and Spatial Range Query

Node Operation When a new node joins the system, it must find a bootstrap node N first. Next, the new node generates a *destination key* by randomly selecting the region bits and the CAN substrate routes it to the zone with the node key value closest to the destination key. When the new node reaches this destination zone, it checks the zone and its immediate neighbors to find a candidate zone to split. Because spatial data objects are usually not uniformly distributed over the complete space, some parts of the space contain a lot of data objects while others manage almost no data. A zone containing many data objects may result in many queries being performed in that area, and hence a performance bottleneck might arise. Consequently, we need to balance the number of data objects in each zone. Moreover, in our design the zone being selected for a new joining node must be larger than the *minimum zone size* ($\frac{1}{2^{(a+b)}}$). As an additional concern, there might exist some very large empty regions. Therefore, checking the destination zone and its immediate neighbors might not suffice to balance the number of data objects in a particular region. Thus, we modified

the node join mechanism of CAN as follows. A new node will choose a zone with the local maximum number of data objects and a size that is larger than the minimum zone size to be split. We also propose a threshold, TH, as an upper bound on the number of search hops to find a zone to split. When the new node reaches the destination area, it checks the number of data objects in this destination zone and its neighbors recursively. If the number of zones surveyed exceeds TH, the procedure is stopped and the node which is larger than the minimum size and contains the most data objects will be split. The node key of the new node will be set to the VID of the zone. The value of TH is based on the degree of locality of the spatial data being stored in the system. If the size of all the zones being checked in this procedure is equal to the minimum zone size, the new node duplicates the archive information of the zone with the largest number of data objects surveyed. In the future, range queries for this area will be routed to either one of these duplicated nodes randomly. However, a data object update mechanism must be implemented and the immediate neighbors of this area need to be informed about this duplication. Finally, node departure and node merging in this design is based on the original CAN mechanism.

Data Object Insertion For any data object I insertion, the originating node checks how many scatter regions overlap with the object. For each overlapped scatter region, the node executes the hash function to generate a data key, H_i , of the object. The scatter region part of these keys is set by the VIDs of the overlapped zones. The region bits of these keys map each one of them deterministically into an existing zone Z of the CAN space. Moreover, there must exist a zone Y , which equals or covers the zone Z and stores H_i . Therefore, H_i can be routed to zone Y and stored there.

Data Object Deletion For the deletion of data object D , the originating node must determine the scatter regions of the object from its spatial properties and send deletion messages to the relevant scatter regions. The message includes a *passed zone list* and the object identifier (i.e., the content hash result) of D . After arriving at the designated scatter regions, the message is forwarded among zones and any passed zones will be stored in the passed zone list to avoid loops. If a data object is found with the same object identifier as D , it will be removed and a confirmation message will be returned to the originating node.

Spatial Range Query When any node receives a spatial range query Q , it calculates how many scatter regions overlap with Q . Next, the query message is forwarded to these overlapping scatter regions. In each overlapped region, query Q is forwarded among zones and any matching result will be returned to the query node. In addition, any passed zone in an overlapped scatter area will be stored in

the passed zone list to avoid routing loops.

4 Conclusions and Future Research Directions

Peer-to-peer environments have generated intense interest in the research community because of their desirable properties for large scale systems. As a result, spatial data management in P2P environments is becoming increasingly important. We have presented a hash function to preserve both spatial locality information and constrained load balance in P2P systems. Specifically, we have integrated our design with content addressable networks by modifying and extending CAN. We have illustrated the practicality of supporting spatial range queries with a constrained load balance. We plan to extend our work through simulations to acquire more accurate information about the trade-off in performance when adjusting the parameters along the spectrum of spatial locality preservation versus data load balance.

References

- [1] FIPS 180-1. Secure Hash Standard. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 1995.
- [2] R. Finkel and J. Bentley. Quadtree: A data structure for retrieval on composite keys. *ACTA Informatica*, 4(1):1–9, 1974.
- [3] A. Harwood and E. Tanin. Hashing Spatial Content over Peer-to-Peer Networks. In *Proceedings of the 2003 Australian Telecommunications, Networks and Applications Conference*, Melbourne, Australia, 2003.
- [4] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content Addressable Network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 27-31, 2001.
- [5] 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.
- [6] 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 the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149–160. ACM Press, 2001.
- [7] S. Zhou, G. R. Ganger, and P. Steenkiste. Location-based Node IDs: Enabling Explicit Locality in DHTs. Technical Report CMU-CS-03-171, School of Computer Science, Carnegie Mellon University, 2003.