A Distributed Quadtree Index for Peer-to-Peer Settings

Conference Paper · May 2005

DOI: 10.1109/ICDE.2005.7 · Source: IEEE Xplore

3 authors, including:

Egemen Tanin
University of Melbourne
164 PUBLICATIONS **2,018** CITATIONS

SEE PROFILE

Hanan Samet
University of Maryland, College Park
371 PUBLICATIONS **17,257** CITATIONS

SEE PROFILE

A Distributed Quadtree Index for Peer-to-Peer Settings*

Egemen Tanin† Aaron Harwood† Hanan Samet‡
†Department of Computer Science and Software Engineering, University of Melbourne
‡Department of Computer Science, University of Maryland at College Park
(www.cs.mu.oz.au/p2p)

1 Distributed Quadtree-based Hashing

For a Peer-to-Peer (P2P) application to provide the functionality readily available on client-server systems, it must be capable of facilitating spatial queries such as "selecting all real-estate ads for properties in a given region in a city", e.g., in a P2P auction network. We introduce a P2P quadtree index to support such accesses. It is based on the *Chord* method [3]. Methods such as Chord have been gaining usage in P2P settings to facilitate exact-match queries. The Chord method maps both the data keys and peer addresses to a common set of virtual locations using a hash function (e.g., SHA-1). In Chord, each peer maintains a table of other peer addresses. Each entry in the table represents an interval that is double the interval size of the previous entry in a circular name space starting from the position of the peer in this space. Each interval is associated with a location to which a peer is already mapped. Given a file name, the tables can then be used to traverse the network and to find data files. In general, given n peers, it can be proven that, with a high probability, a request to locate a file will be forwarded $O(\log n)$ times.

Mondal *et. al* [2] describe preliminary results on indexing spatial data using R-trees and queries on this data for P2P networks, while we are using a quadtree variant. Ganesan et. al [1] describe two approaches to handle range queries on point data that can also be applied to other spatial data sets. First, they use space-filling curves to reduce multi-dimensional data into one dimension and then utilize skip-graphs for range queries. Second, they introduce a P2P version of a kd-tree.

A spatial query can be efficiently executed by recursively subdividing the space into four congruent regions, i.e., finding the result using some variant of a quadtree. With our distributed quadtree index we assign responsibilities for regions of space to the peers. If a peer is responsible for a region of space, then it is responsible for query computations that intersect that region of space and for storing the objects that are associated with that region. Each quadtree block can be uniquely identified by its centroid. We call this centroid a *control point*. These control points are hashed using the Chord method so that the responsibility for a quadtree block is associated with a peer. For example, the result of hash function $H(\text{“}(5,2)\text{”})$ is used as the location of the control point $(5,2)$ on the Chord. The control points can be determined using the globally known quadtree subdivision method to recursively subdivide the space. Given a control point, there is a unique mapping to a quadtree block. With a good base hash function, we can achieve a uniformly random mapping of the quadtree blocks to the peers.

A query starts at the root of the quadtree and is propagated down through the branches of the tree, testing for the intersection with objects as it proceeds. For a P2P system, the tree traversal becomes a set of peer visits. The query is transmitted from a parent block of a quadtree, i.e., from the peer to which the control point maps to the child blocks, i.e., to the peers that store the child blocks, by utilizing the Chord P2P lookup method. Note that there is a single point of failure that occurs had all tree operations are started at the peer that stores the root control point. On the other hand, as we have a distributed structure, there is no reason for us to start all the operations from the root of the tree. Therefore, we introduce the concept of the *fundamental minimum* level, f_{min}. This modification forces objects to be stored and query processing to start at a level $l \geq f_{min}$. Hence, no objects can be stored at levels $0 \leq l < f_{min}$ for our trees. The objects are subdivided into parts and then stored at lower levels. The concept of f_{min} also avoids any potential overloading of peers that would have otherwise stored control points at a level less than f_{min}. We also use f_{max} as the *fundamental maximum* level which allows us to prevent objects from falling deeper than a level that is greater than f_{max}. Values for f_{max} and f_{min} are constant, globally known, and can be chosen per application. With $f_{min} = 0$ our structure reverts to the standard quadtree, but a distributed version of it. With $f_{min} = f_{max}$, our structure degenerates to a distributed grid in which case, the tree structure has been collapsed into a single level. Operations on our index are decentralized and queries are delegated to child-blocks using the Chord method. Chord usually takes $O(\log n)$ messages to reach a destination (n is the number of peers in the system). Since each node of our distributed tree has a fixed number of children, we allow each node to maintain a cache of addresses for its children and thereby reduce the delegation message complexity to $O(1)$. Our caches need updates with incoming/outgoing peers and we

*The support of the National Science Foundation under Grants EIA-99-00268, IIS-00-86162, and EIA-00-91474, and Microsoft Research is gratefully acknowledged.

make our updates when a cache miss occurs.

2 Experimental Results

Our simulation environment used the ns-2 package, an advanced network simulator, in tandem with the GT-ITM package. Our peers were placed randomly on the nodes of a generated transit-stub topology. We created a P2P real-estate application for the Internet. Users were assumed to be making area selections from a map to search for houses or land for sale or rent. Extra data was associated with each spatial object, e.g., a picture and/or a set of descriptive attributes of the house/land for sale. This was made available from the original peer that inserted this spatial object to the system. For our experiments, we submitted a given number of queries, arriving as a flash crowd, to the system.

We first looked at how our index scales with the increasing number of queries and with respect to a standard client-server system. There were 1000 peers in the network hosting 1000 spatial objects and the data attached to these objects were assumed to be 100 KBytes. The f_{min} value was 3 and the f_{max} value was 10. Our cache is assumed to have no misses for this experiment. As an obvious result, our distributed quadtree index shows almost no degradation with the increasing number of queries. The only time that the P2P system has a larger cost was when there were very few (i.e., 10) queries in the system, as it is much easier to go to a single server (that is not congested) to obtain some data. Once the number of queries reached 20, the client-server started to perform poorly. For more than 300 queries, there is an order of magnitude difference between the central and the distributed index. The time needed to download a large number of hits determines the performance when we have a large number of queries.

We also wanted to find how our index scales as the number of peers in the system increases. The number of queries is fixed at 100 for this experiment. The number of objects is 3000. The remaining experiment parameters have the same values as in the first experiment. We used miss ratios of 0 and 5 percent and repeated the experiment twice. We saw almost no change when the number of peers increased from 1000 to 3000. Basically, due to caching, the number of peers loses most of its relevance for the distributed tree, except during the query initiatization period (i.e., finding the peers at level f_{min}). We also observed that for the case where the miss ratio is 5 percent, there is almost no significant change in the behavior of the index.

Our most interesting experiment was the load-balance experiment. This experiment observed the relationship between f_{min} and the load-balance in the system and, more importantly, the overall distribution of load using our index over peers (Fig. 1). The number of queries was fixed at 100 for this experiment (the remaining experiment parameters have the same values as the first experiment). We can see that about 99 percent of all peers process between 0 to 60 messages regardless of the value of f_{min}. With increasing values of f_{min}, we see a gradual improvement in load-balance. For example, there are approximately 807 peers for $f_{min} = 3$ that share the load of up to 30 messages, while

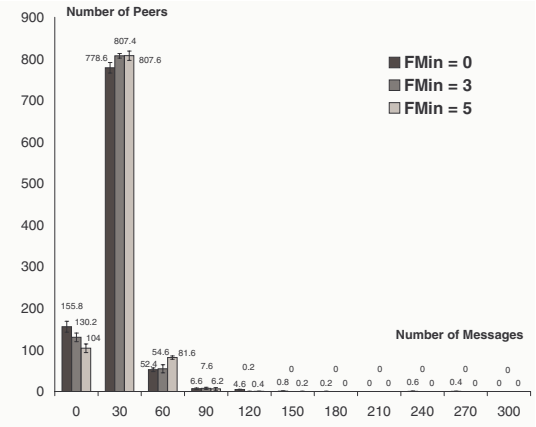

Figure 1. Load-balance for three f_{min} values.

there are 779 peers for $f_{min} = 0$. But, more importantly, the possibility to observe a few peers with a drastically high number of messages diminishes. For example, we can see that for $f_{min} = 0$ there is a possibility for observing a peer with more than 240 messages as the graph has the value of 0.4. On the other hand, for $f_{min} = 5$ or $f_{min} = 3$ there is not even a single run with a peer that handles more than 150 messages, and even the probability of this occurring is very small (for up to 150 messages we see 0.2 on the graph). It is important to note that with $f_{min} = 5$ we see an increase in the number of peers that handle more than 30 messages, many of which are redundant messages as the tree starts to lose its pruning capability since it approaches to a grid.

In summary, we have described a distributed quadtree index for enabling more powerful accesses on complex data over P2P networks. Our work can be applied to higher dimensions, to various data types, i.e., other than spatial data, and to different types of quadtrees. Finally, we can use other key-based methods than the Chord method as our base P2P routing protocol. Our algorithms and index scale well. The index also benefits from the underlying fault-tolerant hashing-based methods by achieving a nice load distribution among many peers. We can seamlessly execute a single query on multiple branches of the index hosted by a dynamic set of peers.

References

[1] P. Ganesan, B. Yang, and H. Garcia-Molina. One torus to rule them all: Multidimensional queries in P2P systems. In *Proceedings of the ACM SIGMOD'04, WebDB Workshop*, pages 19–24, Paris, France, June 2004.

[2] A. Mondal, Yilifu, and M. Kitsuregawa. P2PR-tree: An R-tree-based spatial index for peer-to-peer environments. In *Proceedings of the International Workshop on Peer-to-Peer Computing and Databases (held in conjunction with EDBT)*, Heraklion-Crete, Greece, March 2004.

[3] 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 ACM SIGCOMM'01*, pages 149–160, San Diego, CA, August 2001.