A Serverless 3D World[*]

Egemen Tanin[†] Aaron Harwood[†] Hanan Samet[‡] Sarana Nutanong[†]
Minh Tri Truong[†]

[†]Department of Computer Science and Software Engineering
University of Melbourne
(www.cs.mu.oz.au/p2p)

[‡]Department of Computer Science
Center for Automation Research
Institute for Advanced Computer Studies
University of Maryland at College Park
(hjs@cs.umd.edu)

ABSTRACT

Online multi-participant virtual-world systems have attracted significant interest from the Internet community but are hindered by their inability to efficiently support interactivity for a large number of participants. Current solutions divide a large virtual-world into a few mutually exclusive zones, with each zone controlled by a different server, and/or limit the number of participants per server or per virtual-world. Peer-to-Peer (P2P) systems are known to provide excellent scalability in a networked environment (one peer is introduced to the system by each participant), however current P2P applications can only provide file sharing and other forms of relatively simple data communications. In this paper, we present a generic 3D virtual-world application that runs on a P2P network with no central administration or server. Two issues are addressed by this paper to enable such a spatial application on a P2P network. First, we demonstrate how to index and query a 3D space on a dynamic distributed network. Second, we show how to build such a complex application from the ground level of a P2P routing algorithm. Our work leads to new directions for the development of online virtual-worlds that we believe can be used for many government, industry, and public domain applications.

[*]This work was supported in part by the US National Science Foundation under Grants EIA-99-01636, EIA-99-00268, IIS-00-86162, and EIA-00-91474, and Microsoft Research.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*; C.2.4 [**Computer Communication Networks**]: Distributed Systems—*Distributed applications*; E.1 [**Data Structures**]: Distributed Data Structures

General Terms

Algorithms, Management

Keywords

Virtual-worlds, Spatial data, Peer-to-Peer systems, Distributed hash tables

1. INTRODUCTION

A virtual-world is a representation of objects, along with their relationships in a 3D space (e.g., a virtual-city is a set of roads, buildings, cars, and pedestrians on the same 3D virtual-space). Users can participate in the virtual-world by locating themselves within it, render a view of the representation, and manipulate the objects. Users may themselves be represented in the virtual-world as objects. The use of online virtual-worlds has great potential for government, industry, and in general public domain applications. Recent developments of 3D virtual-world gaming has attracted many participants. Various terrain-modelling based applications on a highly distributed scale can be considered. Many counties over a large geographic area can contribute to a virtual-community with their local data and servers to form a very large distributed virtual-world and processing environment. This data can be harvested by virtual-tourists, scientists, and government employees. Yet, online applications such as multi-participant virtual-world-based games are hindered by their inability to efficiently support a large number of participants on a constant number of servers. Current solutions divide a large virtual-world into mutually exclusive zones, with each zone controlled by a different server, and/or limit the number of participants per server or per virtual-world. These solutions do not scale well because the communication and computation complexity of the system is not distributed and scaled in proportion to the number of participants.

From another perspective, the recent emergence of large file-sharing systems based on the Peer-to-Peer (P2P) paradigm for distributed computing demonstrates how a large number of participants can self-organize into a coherent, global network [6]. In comparison to other applications, where a large number of participants is generally used to mean thousands of users, P2P systems consistently sustained millions of users. A large number of participants in our case is generally used to mean orders of magnitude more than many online applications. Recently, significant amount of papers has focussed on various indices that can be used for accessing complete data files on P2P networks for various purposes. Examples of this work are [1, 4, 14, 16, 17, 18, 20, 21]. Some prototype P2P systems are already built on top of new low-level P2P utilities that use these indices. For example, the PAST [9] persistent storage system is built on the Pastry [18] routing and indexing service. Such systems aim to form the future of data storage and retrieval. Platforms like JXTA are also emerging (www.jxta.org) where people can implement simple P2P systems.

There are many advantages of P2P systems from an online virtual-world application point of view. First, for each participant a new host joins the system. Second, the bandwidth and processing power is not gathered on a few bottleneck servers. Yet, it is not an obvious task to accommodate a virtual-world (or any other spatial) application on a P2P system. Many participants will enter and leave the application along with their hosts. Maintaining a complex virtual-world on such a dynamic environment in a transparent manner is very difficult. Also, current P2P indices cannot provide the necessary functionality to perform many types of queries on 3D data that users would otherwise expect from any regular client-server based system (such as range queries that are crucial for spatial applications). Existing indices are geared towards finding complete objects, like files, given a unique key, i.e., a file name. Finally, it is not obvious how to engineer such a 3D application on a P2P network from the base level of a P2P routing algorithm and using basic constructs such as sockets.

1.1 Our Contribution

In this paper, we present a P2P solution for enabling online virtual-worlds that admit a large number of participants without placing excessive burden on any particular host in the system. In the context of file sharing, P2P systems use an index that maps files to peers on the network. Similarly, the basis of a virtual-world is an efficient index of the objects in the world and an efficient querying system to provide for complex queries on this world. We propose an index for distributing the objects over the peers in a P2P network. If each peer is a participant in the virtual-world then each peer is providing some fraction of the total work required to maintain and process the objects. This relieves the administrators from having to provide central hosts with enough capacity to serve an ever growing number of participants. Ideally, no administration is required in our case.

First, we introduce and analyze an index that is based on octrees and distributed hashing for enabling more powerful accesses on 3D spatial data over P2P networks. Our work, like some of the previous related work, relies on creating a globally known mapping between the node addresses of a P2P network and the data that will be available in this network. To avoid all-to-all communications, the mapping function has to be known by all the peers of the network. Second, we outline a layered architecture, which we call Open P2P Network (OPeN) architecture, for the design and implementation of complex P2P applications such as our 3D virtual-world application. The OPeN architecture provides an explicit object-oriented solution that delegates data processing over a P2P network, rather than forcing all processing of data to be centralized. Also, P2P applications are separated from the P2P routing protocol. Applications can be developed by using various simple core services. The OPeN architecture ensures that applications adhere to the P2P paradigm so that they are intrinsically decentralized and autonomous.

2. DISTRIBUTION OF THE VIRTUAL-WORLD

We use peers in a P2P network to store objects of the virtual-world. Each peer is assigned the responsibility for some regions of the virtual-world and any objects that are within that region. The regions are defined using a recursive octree subdivision of the space and objects are placed into the smallest regions which contain them entirely. Basically, we distribute an octree over a P2P network in a manner that ensures load balancing between peers.

2.1 Distributed Hashing

Hashing is becoming increasingly popular for mapping and accessing distributed data over large networks (e.g., [11]). Although there are many methods for a distributed system to implement a distributed hash algorithm [14], we use a method that has recently become widely known as the *Chord* method [20]. Our work is built upon the Chord method although we believe that other key-based methods can also be used. The Chord method, and hence our work, can accommodate a large number of dynamic nodes in a distributed environment. In the simplest sense, the Chord method can be viewed as a routing protocol to find a file given its name in a P2P network.

Distributed hashing uses a hash function to map arbitrary data strings, i.e., keys, onto a logical space. Some spaces are multi-dimensional and some spaces are one-dimensional, e.g., as used in Chord. In both cases, each peer is responsible for maintaining some subset of the logical space. Also, each peer's Internet address is hashed onto this space, called the peer's location, and then the peer is responsible for some subset of the logical space that is nearest to that location (defined by an ordered relation). Plus, each peer records the IP addresses (note that port numbers are also needed but omitted for the simplicity of this discussion) of some of the peers in the logical space. This connectivity holds the P2P system together. For a d-dimensional space this whole design can form a d-dimensional torus (assuming peers know only about their closest neighbors and we use modulo arithmetic). So, in one dimension, the P2P network is simply a circle. Fig. 1 depicts the Chord design that uses a circle and also shows other details from the Chord method.

Each peer in the Chord maintains a table of up to t other peers, where t is logarithmically proportional to the number

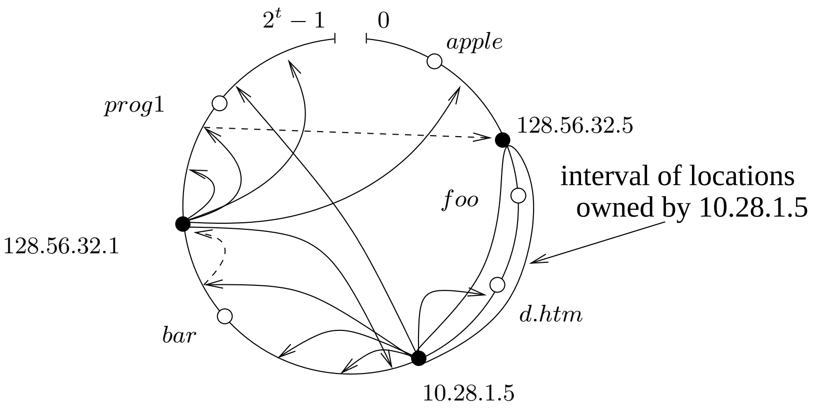

Figure 1: The Chord method.

of locations. Each entry in the table represents an interval that is larger than the interval of the previous entry (usually twice as large). In the example in Fig. 1, the entries in the table for two peers are shown as arrows from the peers. Each interval is associated with a successor peer (the owner of the corresponding entry). These are depicted using directed broken lines although only a subset relevant to this example are actually displayed. These table entries are then used to jump from one peer to another, towards the peer that maintains the data file that the user is looking for. After completing all the tables for all the peers in the network, one can see that the circled area in the figure shows the name-space/locations for which peer 10.28.1.5 is responsible for. Similarly, each peer takes over the responsibility of a region in this space. Any file with a name that falls into a region will be owned by a designated peer. Note that the formation of the division of space and the table entries for each peer do not require an all-to-all communication.

Without giving an explicit algorithm, consider the case when peer 10.28.1.5 is trying to locate the object with key *prog1*. Hence, we are looking for the peer address who has *prog1* so that we can contact that peer and get *prog1* itself. We will now use our table of peers to trace the location of *prog1*. The querying peer 10.28.1.5 checks to see if he owns *prog1* and if not it scans through its table and finds the name space interval to which the key *prog1* maps. In this case, *prog1* is in the interval defined by the third and fourth arrows, counting clockwise from 10.28.1.5, mapping to our third and fourth entries in the peer table. We contact the peer that is the successor peer for the third arrow. Note that using the fourth arrow rather than the third arrow may miss a peer that may exist between *prog1* and the fourth arrow. The request for *prog1* is now forwarded to peer 128.56.32.1, getting us closer to the destination. Now, 128.56.32.1 checks whether it has *prog1* and if not (which is the case here as *prog1* is between the second and third interval defined by the arrows from peer 128.56.32.1), then it repeats the process and hence forwards the request to peer 128.56.32.5 as the broken arrow from the end of the second interval is directed to it. This is the peer that knows who actually has the *prog1* itself. For this example, we assumed that objects are not relocated with keys when they were first hashed (i.e., in cases where large data files are stored in the P2P system that we do not want to move around). In general, it can be proven that a request to locate an object/data file will be forwarded $O(\log n)$ times with a high probability, where n is the total number of peers in such an application of the Chord method. The Chord method is shown to be very resilient to peers leaving and coming to the system. In general, the Chord [20] method has many other useful properties that makes it a suitable routing algorithm for P2P systems that are not mentioned in this paper.

2.2 Distribution of Objects and Queries

Objects need to be inserted/deleted/modified within the virtual-world and queries are required to compute which objects are within a small region of the virtual-world. A query can also be defined as a spatial object and the result of the query consists of all the objects that intersect with the query. Unfortunately, 3D objects do not have names like files and hence a simple adaptation of the Chord method fails.

Using a simple multi-dimensional 3D logical space and mapping regions of the 3D virtual-world to this space and also hashing peer addresses to the same simple logical space may seem favourable for this application. Each point in the virtual-world is thereby assigned ownership to some peer, or in reverse, each peer has a location in the virtual-world and owns the region of space that is nearest to it. This method may work well when the objects in the space are uniformly distributed over the space. However we have to realistically consider non-uniform distributions of objects for a 3D virtual-world. So, when peers are distributed over the space uniformly at random using a hash function, this means that using a one-to-one mapping from the virtual-world to the logical space can lead to some peers receiving a greater number of objects than others when objects are non-uniformly distributed.

To avoid load balancing problems arising from non-uniform distributions of objects, and due to the fact that objects in space do not have names that we can immediately use for hashing, we distribute nodes of an octree that subdivides the virtual-world onto the one-dimensional Chord logical space. Any good hash function with a uniformly random distribution of keys to locations can be used to take the center point of the region of the related tree node, form a string out of its coordinates, and map it onto the Chord (i.e., SHA-1 can be used as the base hash function, www.itl.nist.gov/fipspubs/fip180-1.htm). This means that for any given node of the tree that represents some region of the virtual-world, it will be assigned to a peer selected uniformly at random. The assignment provides a good load balancing that counters non-uniform object dis-

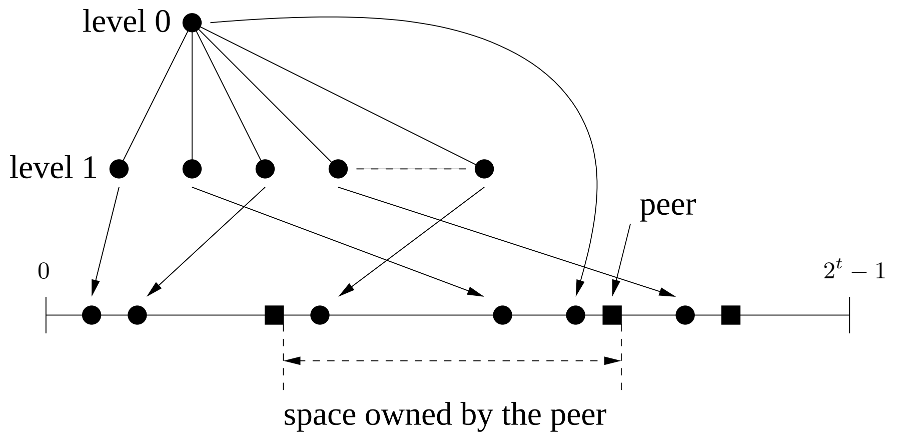

Figure 2: The first two levels of an octree mapped onto the logical space; and 3 peers that are responsible for different subsets of this space.

tributions assuming that the octree subdivisions continue to a deep enough level for each dense region of the world. Fig. 2 shows level 0 and level 1 of an octree and an example mapping of the nodes onto a logical space of 2^t identifiers. Typically, $t = 160$ for many P2P applications. Each peer (the squares) is responsible for the elements of the logical space that come before it (inclusive) and the previous peer.

In $\mathbb{R}^3$ space, a query can be efficiently executed by using a variant of the octree concept. The objects are indexed into the octree using some predefined subdivision rule. This has the effect of exponentially reducing the average number of intersection calculations per query. For example, using a suitably defined representation, each descent to a new level of the octree reduces the number of possible resulting objects by a factor of eight. After our mapping of tree nodes to the logical Chord space, we can see that if a peer is responsible for a region of space, then it is responsible for query computations that intersect that region of space, i.e., with the objects that are associated with that region. We are now capable of answering any spatial query on a P2P system (including range queries over objects that span a 3D space).

In our octrees, each object is associated with the smallest octree block that contains the object in its entirety. Subdivision ceases whenever an octree block contains no objects. This is a three-dimensional variant of the MX-CIF quadtree [2, 12, 19]. For each subdivision, there is a center point, termed a *control point*, that the subdivision planes of the underlying space intersect. Specifically, we hash these control points so that the responsibility for an octree-block is associated with a peer in the P2P system. For example, $H(\text{“}(5,2,7)\text{”})$ is the location of the control point $(5,2,7)$. We allow the control points to be dynamically determined using a globally known function to recursively subdivide space. Hence, each peer is made responsible for some regions of space using the control points that hash to that peer. A control point is used in our algorithm like a bucket for storage of objects and also performing intersection calculations associated with that region. Given a control point, there is a unique mapping to an octree block.

2.3 Distributed Spatial Algorithms

We further modify the octree concept with an alteration that forces objects to be stored and query processing to start at a level $l \geq f_{min}$ where f_{min} is said to be the *fundamental minimum* level, i.e., no objects can be stored at levels $0, 1, \ldots, f_{min} - 1$. We also use f_{max} as the *fundamental maximum* level and this allows us to prevent objects from falling lower than a level that is greater than f_{max}. Values for f_{max} and f_{min} are constant and globally known. With $f_{min} = 0$ our structure reverts to a simple octree. With $f_{min} = f_{max}$ our structure degenerates to a 3D mesh; completely collapsing the tree structure into a single level. The use of f_{min} removes the single point of failure that would have occurred had all tree operations begun at the peer that stores the root control point. It also helps further balance the load and avoid any potential overloading of peers that would have otherwise stored control points at a level less than f_{min}.

Nodes of the tree are distributed in a uniformly random manner over the peers using the base hash function of the Chord method. Hence, we distribute the tree operations, *insert*, *query*, and *delete*, by branching and *delegating* control to the peers that are responsible for the control points from the octree. Parts of an operation can work in parallel on different branches of the distributed octree as they map to different peers through different control points.

Each control point u has the following data structure associated with it:

$$D(u) = (\{d_1, d_2, \ldots, d_8\}, \mathit{list}).$$

Downward counts $d_i \in \mathbb{N}$ are used to indicate the number of objects that do exist at or below child i. The *list* is a list of objects that intersect the region $R(u)$ and that could not fall any further down the tree. The default values for $D(u)$ are:

$$\{\overbrace{0, 0, \ldots, 0}^{8}\}, \mathit{empty}$$

which means that there are no objects below u and no objects stored at u.

A peer calls $\text{InsertObject}(\text{object } X)$ to insert an object into the tree. We use $\text{Delegate}(u) \rightarrow \text{Func}()$ to mean that a peer sends a control message that invokes $\text{Func}()$ on another peer that stores control point u. A $\text{Delegate}()$ operation is similar to creating a new thread of control. Procedure $\text{InsertObject}()$, shown below, concurrently delegates procedure $\text{DoInsert}()$ over all control points at level f_{min} that intersect with the object. Procedure DoInsert() delegates recursively and also concurrently through the distributed tree until the object is inserted. Note that $D(u).field$ is used to access/set $field$ of/for $D(u)$. In our presentation, we make use of the following auxiliary procedures and notation. $Ints(X, Y)$ computes the intersection of X with Y. $R(u) = (x_1, y_1, z_1, x_2, y_2, z_2)$ denotes the region defined (and also controlled) by control point $u = \left(\frac{x_2+x_1}{2}, \frac{y_2+y_1}{2}, \frac{z_2+z_1}{2}\right)$. $L(u)$ denotes the level of control point u. $C(u, i)$ represents the i-th child of control point u, where $i = 1, 2, \ldots, 8$.

```
InsertObject(object $X$)
{
  control point list $G$:={}
  Subdivide($X, root, G$)
  **for** each $u$ in $G$ **do**
    Delegate($u$)→DoInsert($X$,$u$)
}

DoInsert(object $X$,
         control point $u$)
{
  if ($X$ is not within exactly one $R(C(u,i))$)
   or ($L(u) = f_{max}$) then
   set $D(u)$.$list$ to include $X$
  else
    for $i$:=1 to 8 do
      if ($Ints(X, R(C(u,i)))$ is not empty) then
       increment $D(u)$.$d_i$ by 1
       Delegate($C(u,i)$)→DoInsert($X$,$C(u,i)$)
}
```

Procedure Subdivide($X, root, G$) is called with initially $G = \{\}$ when inserting X (also used when deleting or starting a query). This initial subdivision of an object (or query) down to level f_{min} is performed by the recursive progress of Subdivide(): The procedure modifies G by adding control points to it. First, $root$ is used to call this method where $L(root) = 0$ and $R(root)$ is a bounding box that bounds all data and query objects. For this algorithm it is assumed that all X's will be contained within $R(root)$. The list of control points at level f_{min}, G, is computed locally and processing is then delegated to the peers that store these control points.

If DoInsert() is invoked on a control point that does not exist then the control point is implicitly allocated with default parameters. Delegation is sent using the Chord method and so ordinarily it takes $O(\log n)$ messages to reach its destination (n is the number of peers in the system). Since each node of the 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 $O(1)$ (as we will no longer need to use the traversal algorithm of Chord for each child). This is true only when stepping through the tree. The number of peers that are initially contacted is a function of f_{min} and can consist of a large number of addresses. Hence, we do not allow caching of the f_{min} level peers' addresses.

```
Subdivide(object $X$,
          control point $u$,
          control point list $G$)
{
  if ($L(u) = f_{min}$) then
   add $u$ to $G$
   return
  for each child $v$:=$C(u,i)$ do
    if ($Ints(R(v), X)$ is not empty) then
     Subdivide($X, v, G$)
}
```

Procedures for DeleteObject() and DoDelete() are almost identical to InsertObject() and DoInsert() and so an explicit listing is not given here. The essential difference is that objects are removed from $D(u).list$ instead of being added to it and that $D(u).d_i$ is decremented instead of being incremented by 1.

Peers can receive a query from any node on the Internet (i.e., a client may or may not be a peer in the system) via the ReceiveClientsQuery() procedure. Similar to insertion (deletion), this procedure takes in an object (named Q for query in this case) and then it finds the control points at level f_{min} and delegates the query to the relevant peers. The results of a query can then be sent back to the client.

We use the term hit to indicate that an object intersects a query. A query which covers multiple control points may return the same object a multiple number of times because some objects may have been copied into multiple control points when forced down to a level at or below f_{min}. Thus we have to eliminate such superfluous hits, i.e., same object intersecting with the same query multiple times at different peers concurrently.

```
ReceiveClientsQuery(query $Q$)
{
  control point list $G$:={}
  Subdivide($Q, root, G$)
  for each $u$ in $G$ do
    Delegate($u$)→DoQuery($Q$,$u$)
}

DoQuery(query $Q$,
        control point $u$)
{
  intersect objects in $D(u)$.$list$ with $Q$
  send results to client
  for $i$:=1 to 8 do
    if ($Ints(R(C(u,i)), Q)$ is not empty)
       and ($D(u)$.$d_i > 0$) then
     Delegate($C(u,i)$)→DoQuery($Q$,$C(u,i)$)
}
```

2.4 Complexity Analysis

For queries, there are two main operations that affect the performance of the system: *(i)* downloading objects from peers to the clients and *(ii)* sending query and control messages between peers or from clients to peers to find the hits. We consider computation overheads to be negligible and hence network operations are assumed to be the main source of delay. We let the function $t_d(s)$ denote the time to transmit some data of size s, and let the constant t_m denote the time to transmit a single control message.

For a client-server system that simultaneously receives q queries with h hits per query, with each hit having some data of size b bytes, the bottleneck will be the server connection. The time for the client-server system to answer the queries is:

$$T_{cs}(q) = t_m + t_d(q.h.b) \quad (1)$$

We assume that the initial query messages are too small to form the bottleneck in comparison to the hits for this analysis.

Let the maximum number of nodes in an octree of height l be:

$$N(l) = \frac{1}{7}.\left(8^{1+l} - 1\right) \tag{2}$$

If we consider the common case when c objects are distributed uniformly at random over n buckets, i.e., peers. It can be shown that the average number of buckets that will receive any objects is:

$$\phi(n, c) = n - ((n-1)^c.n^{1-c}) \tag{3}$$

The intuition behind this formula is that the probability that a bucket receives none of the c objects is $((n-1)/n)^c$.

We assume that the octree is distributed over the P2P network with each node of the tree placed uniformly at random onto a peer. So, for example, if there are $k.N(f_{max})$ octree nodes, where $0 < k \leq 1$ is the fraction of nodes actually used, then an average of $\phi(n, k.N(f_{max}))$ peers will store at least one node.

The time to complete q queries for the P2P network, $T_{p2p}(q)$, is the sum of the time to find the hits and the time to download the hits. Whether the download bottleneck is at the serving peers or at the client depends on a number of factors. For q queries, if h hits per query and data sizes of b is used again, then the load per peer that contains at least one hit is:

$$\begin{aligned}\frac{q.h.b}{\phi(n, q.h)} &= \frac{q.h.b}{n - ((n-1)^{q.h}.n^{1-(q.h)}} \\ &= b \text{ for } n \gg q.h\end{aligned} \tag{4}$$

Note that for a very large number of peers in comparison to the number of hits and queries, there will probably be no more than a single hit per peer (which is of size b bytes) and there may be many idle peers that do not serve any hits in this case.

However the load at the client, which is receiving the load of a single query, is $(h.b)$. The client is receiving objects in parallel from up to h peers. When the client's download time exceeds the peers' upload time then the bottleneck for this data transfer is at the client; otherwise, it is at the peers.

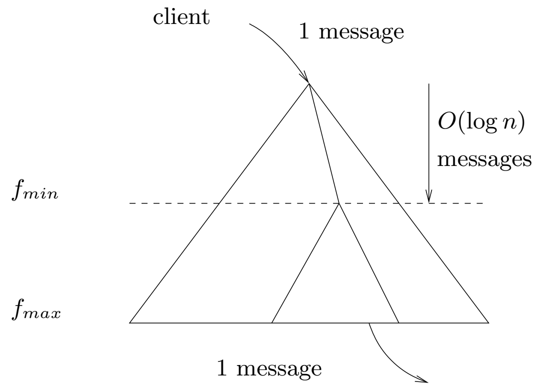

Figure 3: A query starts from f_{min} and propagates to f_{max}.

The propagation of the query is shown in Fig. 3 and following the diagram we can obtain:

$$\begin{aligned}T_{p2p}(q) = &((2 + f_{max} - f_{min} + (\alpha. \log_2 n)).t_m) \\ &+ \max\left\{t_d(h.b), t_d\left(\frac{(q.h.b)}{\phi(n, q.h)}\right)\right\}\end{aligned} \tag{5}$$

The first part of the formula represents the traversal of the distributed tree. The second part represents the download of the hits. The message at the bottom of the tree in Fig. 3 is, for our implementations, due to the last peer sending a message to the peer that stores and is the origin of the data for the hit. Hence, it can send extra information, such as texture, about the object to the client. In reality, $\alpha = 0.5$ is observed for Chord [20]. This value shows the portion of the number of links that will be traversed in a real-life situation for a lookup in the circular Chord name space. In general, we do not expect a spatial query that locates some objects in a 3D space to take more time with our P2P index than a regular central octree-based index. Yet, we will have the advantage of distributing the downloads to multiple peers (rather than a single server) and hence the system will scale much better than a client-server system. We are currently working on a simulation based environment to test our work.

2.5 Related Indexing Work

Recently, approaches targeting mainly range queries on P2P networks have started to emerge [5, 7]. For example, PePer [7] supports range queries on one dimension by dividing the space into regular intervals. A closely related approach called MAAN [5] uses locality preserved hashing to map a range of data space to Chord. This approach uses either a direct mapping of the data domain (i.e., a line segment for one dimensional data) to the Chord space or assumes that the input data range and distribution are known in advance to create a more balanced mapping.

In addition, various approaches (e.g., [3]) have been proposed using the CAN (Content-Addressable Network) [17] approach as a base. CAN is also a key-based lookup service like Chord. Most of these approaches rely on the direct continuous mapping of the data space onto the CAN. CAN uses a multi-dimensional Cartesian space. For example, a city can be mapped onto a two dimensional map and then to a two dimensional name space with CAN. In general, such continuous mappings can have load-balancing issues.

Other related work includes [13] which uses directed acyclic graphs to create a range addressable topology and [10] which uses range-partitioning with online balancing algorithms to get around load-balancing problems that can occur in various mappings. In general, none of these approaches consider the case of more complex data such as 3D spatial data and complex queries on this data.

Very recently, Mondal *et. al* in [15] describe their preliminary work whose goal is to accommodate spatial data and queries on this data for P2P networks. To the best of our knowledge, this work and ours form the first pieces of work on spatial data, for P2P networks. In comparison to [15], we are using octrees while they are using R-trees.

In parallel to these efforts, recent work on sensor and ad hoc networks have also contributed useful ideas to our problem domain. For example, [8] introduces an R-tree index in sensor networks. Their work uses explicit cluster leaders for maintaining connectivity while in our work we try to decentralize this concept and maintain connectivity more implicitly. This conforms with the crucial decentralized nature of P2P networks. They also mainly focus on performing nearest neighbor queries in comparison to other general spatial queries.

3. DEVELOPING A 3D VIRTUAL-WORLD ON A P2P NETWORK

Fig. 4 shows a sample generic P2P virtual-world application that uses our algorithms, built in our labs (www.cs.mu.oz.au/p2p). There are two peers and hence two application windows in this figure. Each peer designed and inserted an object (on the top right corner of each application window) to the virtual-world. The boundaries of the virtual-world are shown on the bottom right corners of the applications as an overview. Two small cubes can freely be placed on this overview to define a query region on the whole space (as shown in the figure). The objects in the query regions are then returned to the large center canvases after the distributed search/query operations complete. Each peer sees the same scene from a different view.

It is not obvious how even such a generic virtual-world application can be implemented on a dynamic network. Hence, we defined and used a layered architecture, called an Open P2P Network (OPeN) architecture, for designing and implementing complex applications such as virtual-worlds on P2P networks. Applications are developed without a need to consider P2P protocols and vice versa; P2P protocols are interchangeable without changing the application semantics. The OPeN architecture provides an explicit object-oriented solution to delegate data processing over a P2P network, rather than forcing all processing of data to be centralized at the peer on which the application executes. Applications can be developed by using various common primitives. This makes application development simpler. The OPeN architecture ensures that applications adhere to the P2P paradigm so that they are intrinsically decentralized and autonomous.

The OPeN architecture consists of three layers; depicted in Fig. 5 as the *Application* layer, *Core Services* layer, and *Connectivity* layer. The Core Services layer ensures consistency and easy development for a large range of applications. The Connectivity layer enables P2P protocols to be developed transparently.

3.1 Application Layer

Applications define and allocate objects. More specifically, objects contain data and methods. The use of objects allow applications to *delegate* processing to the peers that store the objects and thus eliminates a crucial deficiency of existing P2P applications that require all data to be first copied to the peer that is executing the application. Applications can invoke methods of objects and objects can execute autonomously to invoke methods of other objects, i.e., objects can interact with each other. This is fundamentally a distributed object architecture, where, in our case, the spatial properties of the objects are used to locate the objects in the P2P network.

Fig. 5 proposes a number of example complex applications in the Application layer. General classes of applications are shown in parentheses. For example, an air-traffic controller system is a class of virtual-world. The software design may define air-traffic control towers as interactive peers that run a P2P application used by air-traffic controller personnel; and aircraft as application objects. Aircraft are autonomous, they can move through the virtual-world and can interact with other entities in this world such as turbulence. The processing load is distributed over the many control tower hosts. Indexing and implementing moving objects on a P2P network is our current research topic.

3.2 Core Services Layer

The Core Services (CS) layer provides a variety of flexible services that support applications. The example CSes in Fig. 5 are *Peer Management*, *Database*, *Virtual Machine*, *Naming*, and *Security*. CSes are built on top of a Base Core Service, also shown in Fig. 5 and make use of a Base Object. The Base CS and Object are used to ensure all CSes are implemented with a consistent view of the underlying P2P networking operations. The CSes shown in Fig. 5 are not meant to be an exhaustive list. New CSes can be added if it is found that existing CSes are inappropriate for the required application support. We list these CSes as examples of key CSes that either cover a large variety of complex applications or are essential to include for most applications. Applications can use the interfaces of the CSes for easy development. Existing standards can be preserved, i.e., ODBC. CSes can also utilize delegation to handle their tasks. For example, a query can spread across many peers without central control. In our case, we implemented a simple CS to insert/delete spatial objects to a 3D space. The generic virtual-world application is built using this simple interface. The service itself implements the distributed octree index but it uses the Chord method implementation at the Connectivity layer to store objects on control points. A query is a spatial object that can traverse the tree recursively and visit control points (hence peers) to find intersecting spatial objects. The Chord method can be replaced with the CAN method without changing the higher layers.

3.3 Connectivity layer

The Connectivity layer serves the purpose of separating the P2P protocol from the rest of the architecture. Different protocols (e.g., Chord) provide different qualities and it is unlikely that a single protocol can efficiently provide all of the qualities. The P2P Object Management sub-layer provides the basis for a complete object-oriented approach to P2P applications. Objects are used to hide all of the underlying activity. A Base Object is used to represent the simplest kind of object that all of the underlying P2P protocols can manipulate. A Base Core Service is used to provide all CSes with a consistent approach to accessing the P2P network. Remote Method Invocation is used for applications to interact with objects and for objects to interact with other objects. This abstraction allows programmers to make use of familiar object-oriented techniques. The separation provided by the P2P Routing Protocol and P2P Object Management sub-layers allows protocols to be changed without requiring changes to occur at the higher layers. It furthermore allows protocols to be bridged.

4. CONCLUSIONS AND FUTURE WORK

The use of online virtual-worlds has great potential for government, industry, and in general public domain applications. Yet, online applications such as multi-participant virtual-world-based games are hindered by their inability to efficiently support a large number of participants on a small number of servers. On the other hand, P2P networks are becoming a common form of scalable online data exchange. But in P2P networks users cannot perform many types of queries on complex data, such as 3D data, and it is not obvious how to build a complex application such as

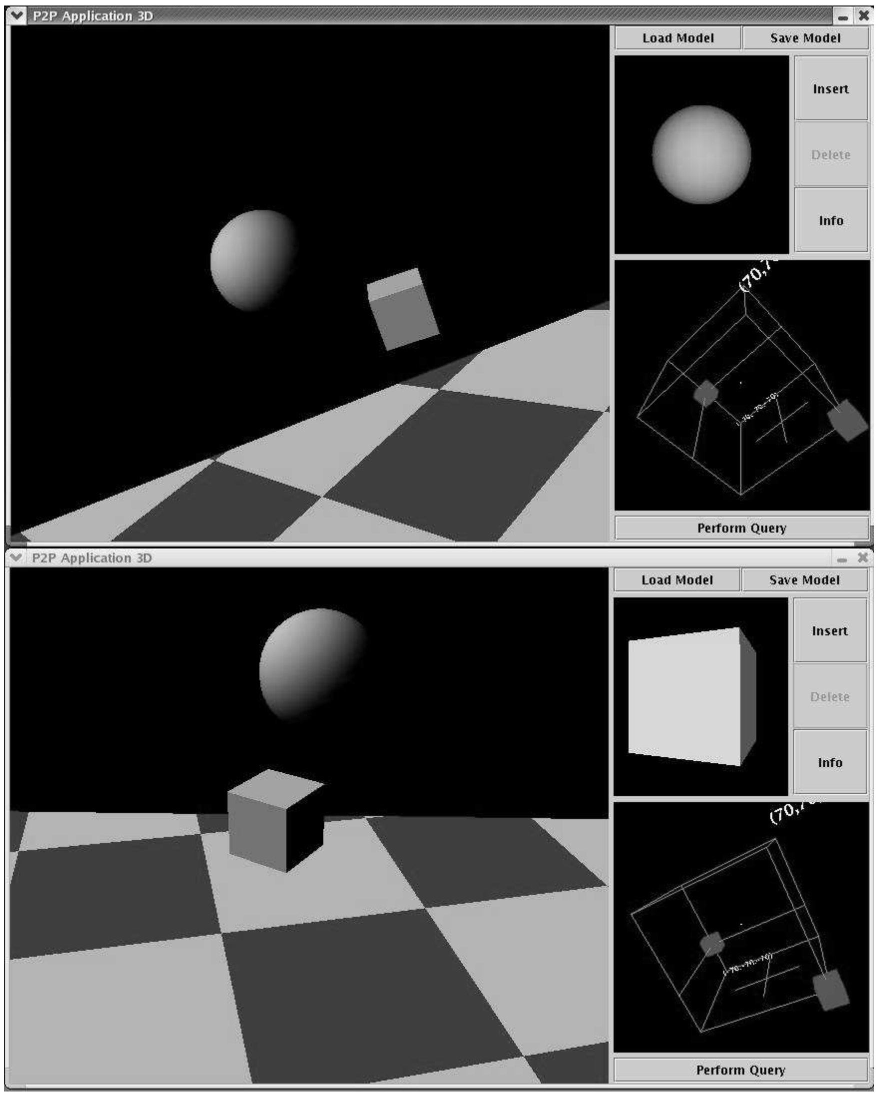

Figure 4: Screen shots of two peers, each viewing the same scene but from different view points. The sphere and cube are different objects in the virtual-world inserted by different peers.

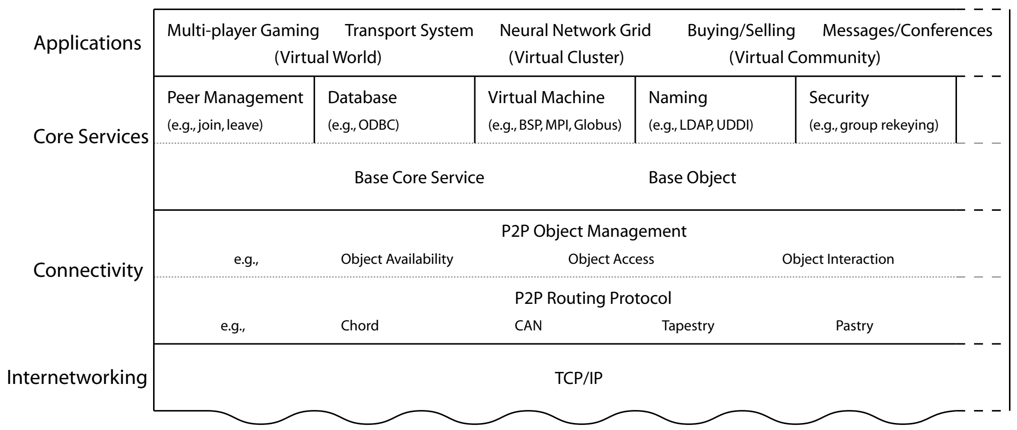

Figure 5: OPeN layered architecture.

3D virtual-world on a P2P network easily. In this paper, we introduced and analyzed a distributed octree-based hashing algorithm and index for enabling more powerful accesses on 3D spatial data over P2P networks. We also showed an architecture for building complex 3D applications over such dynamic networks. Basically, we displayed how a generic scalable 3D virtual-world can be implemented without a server. We believe our work can be applied to various multi-dimensional spaces and using other methods than the Chord method. We are currently experimenting with our methods to observe their performance and usability. We also aim to facilitate moving objects and frequent autonomous object interactions with our work.

Acknowledgements

We wish to thank Deepa Nayar and Jerry Chang for their assistance in developing the implementations.

5. REFERENCES

[1] K. Aberer and M. Punceva. Efficient search in structured peer-to-peer systems: Binary v.s. k-ary unbalanced tree structures. In *Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (held in conjunction with VLDB)*, Berlin, Germany, September 2003.

[2] A. Aboulnaga and J. F. Naughton. Accurate estimation of the cost of spatial selections. In *Proceedings of the 16th IEEE International Conference on Data Engineering*, pages 123–134, San Diego, CA, February 2000.

[3] A. Andrzejak and Z. Xu. Scalable, efficient range queries for Grid information services. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing*, pages 33–40, Linkoping, Sweden, September 2002.

[4] S. Bhattacharjee, P. Keleher, and B. Silaghi. The design of TerraDir. Technical Report CS-TR-4299, Department of Computer Science, University of Maryland at College Park, October 2001.

[5] M. Cai, M. Frank, J. Chen, and P. Szekely. MAAN: A multi-attribute addressable network for Grid information services. In *Proceedings of the International Workshop on Grid Computing*, page 184, Phoenix, AZ, November 2003.

[6] D. Clark. Face-to-face with peer-to-peer networking. *IEEE Computer*, 34:18–21, January 2001.

[7] A. Daskos, S. Ghandeharizadeh, and X. An. PePeR: A distributed range addressing space for P2P systems. In *Proceedings of the International Workshop on Databases, Information Systems, and Peer-to-Peer Computing (held in conjunction with VLDB)*, pages 200–218, Berlin, Germany, September 2003.

[8] M. Demirbas and H. Ferhatosmanoglu. Peer-to-peer spatial queries in sensor networks. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing*, pages 32–39, Linkoping, Sweden, September 2003.

[9] P. Druschel and A. Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the IEEE HotOS VIII Workshop*, pages 65–70, Schloss Elmau, Germany, May 2001.

[10] P. Ganesan, M. Bawa, and H. Garcia-Molina. Online balancing of range-partitioned data with applications to peer-to-peer systems. In *Proceedings of the VLDB*, Toronto, Canada, August 2004.

[11] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigraphy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the ACM Symposium on Theory of Computing*, pages 654–663, El Paso, TX, May 1997.

[12] G. Kedem. The quad-CIF tree: a data structure for hierarchical on-line algorithms. In *Proceedings of the 19th Design Automation Conference*, pages 352–357, Las Vegas, NV, June 1982.

[13] A. Kothari, D. Agrawal, A. Gupta, and S. Suri. Range addressable network: A P2P cache architecture for data ranges. In *Proceedings of the IEEE International Conference on Peer-to-Peer Computing*, pages 14–22, Linkoping, Sweden, September 2003.

[14] W. Litwin and T. Risch. LH*g: A high-availability scalable distributed data structure by record grouping. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):923–927, July 2002.

[15] 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, Greece, March 2004.

[16] C. G. Plaxton, R. Rajaraman, and A. W. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32(3):241–280, May 1999.

[17] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of the ACM SIGCOMM'01*, pages 161–172, San Diego, CA, August 2001.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the ACM Middleware'01*, pages 329–350, Heidelberg, Germany, November 2001.

[19] K. Sevcik and N. Koudas. Filter trees for managing spatial data over a range of size granularities. In *Proceedings of the International Conference on Very Large Data Bases-VLDB*, pages 16–27, Mumbai, India, September 1996.

[20] 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.

[21] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-resilient wide-area location and routing. Technical Report UCB-CSD-01-1141, Department of Computer Science, University of California, Berkeley, April 2001.