Using N-Trees for Scalable Peer-to-Peer Event Ordering

Chris GauthierDickey and Virginia Lo
Department of Computer Science
1202 University of Oregon
Eugene, OR 97403-1202
chrisg | lo@cs.uoregon.edu

Daniel Zappala
Computer Science Department
Brigham Young University
Provo, UT 84602-6576
zappala@cs.byu.edu

***Abstract*— We are concerned with the fundamental problem of event ordering in a peer-to-peer network. A broad set of applications such as distributed simulation, multiplayer networked games, distributed file systems, network monitoring, and peer-to-peer computing need this functionality, but existing peer-to-peer structures (Gnutella, DHTs) cannot provide scalable event ordering. We have developed an event scoping mechanism for scalable event ordering using N-Trees. N-Trees have the advantage of organizing peers so that only those peers involved in a small area of the application space have to exchange event information. This reduces the amount of messaging between peers significantly. Furthermore, N-Trees can handle churn well since *joins* take $O(\lg n)$ time while *leaves* only take a constant amount of time.**

I. INTRODUCTION

Event ordering is a fundamental problem in distributed systems because nodes in a distributed application typically share and modify a common application state space. Without event ordering, the system cannot be correctly reasoned about and will quickly become inconsistent. A broad set of applications require event ordering, including distributed simulation, multiplayer networked games, distributed file systems and databases, network monitoring, e-commerce, and peer-to-peer computing. All of these applications, once moved from an environment of specialized hardware to peer-to-peer networks, are limited by the capacity of the peer-to-peer network to exchange and order events.

The messaging requirements for event ordering limit the scalability of these applications. If Gnutella [1] is a typical peer-to-peer network, then peers will number in the thousands, network connections will be dynamic and may be slow, and peers may join and leave at a rapid rate. Given n peers, each generating m events, using Lamport's strong event ordering would require $O(n^2m)$ messages since each generated event must be sent to every peer [2]. Even weak event ordering has the same messaging complexity since every event which is not local to a peer has to be sent to every other peer [3].

Existing peer-to-peer structures, such as Gnutella, Chord, CAN and Pastry [1], [4], [5], [6], have become a standard peer-to-peer network research topic over the last few years. Various papers detail more efficient designs, improved searching techniques, and build overlays on top of them for efficient group communication (e.g. multicast). However, these structures have not been designed for event ordering and hence will perform poorly for this use.

To address this problem, we propose using N-Trees and *scoped events*. N-Trees are a generalization of *octrees* from computer graphics that recursively subdivide an N-dimensional space [7]. For event ordering, we use an N-Tree to subdivide the *application state space*, which is the state in a peer-to-peer application that all peers share and modify.

Peers are organized into an N-Tree by their location in the application state space. This location is determined by what part of the application state space a peer is interested in reading or writing to. The N-Tree allows peers to quickly and efficiently move and relocate to other parts of the application state space, discover other peers in close proximity, and propagate events to other nodes in the N-Tree.

Peers generate scoped events, which are events that are labelled with a tuple that represents the application state *subspace* that the event modifies. When a peer generates an event, it uses the N-Tree to propagate the event to the other peers within the event's scope. By using scoped

Chris GauthierDickey is supported by a National Science Foundation Graduate Research Fellowship.
Research sponsored in part by NSF ANI 9977524.

events, we can loosen the event ordering requirements in a system from a total ordering of events in the entire system to a total ordering of events in localized areas in the application state space. In other words, event scoping allows us to reduce communication overhead while keeping the system coherent.

This paper has four important contributions. First, we have created a peer-to-peer system for event ordering based on an N-Tree and scoped events, including operations for peers to join and leave the system. Second, we provide an analysis of the complexity of join and leaves in an N-Tree. Third, we show how the N-Tree organization helps reduce the amount of communication between peers, provides fast joining and leaving, and can recover from failures. Last, we show how N-Trees can be used in a real application.

II. Motivation and Background

We can broadly define event ordering into two categories: *strong event ordering*, defined by Lamport as having all events in the system totally ordered [2], and *weak event* ordering, in which only events that do not occur locally in a node are totally ordered (such as accesses to shared memory)[3]. For every event generated by a node that does affects the shared application space, a message must be sent to every other node.

In both types of event ordering, the distributed system must compensate for events that arrive late at other nodes even though they occurred previously in time. Jefferson's Time Warp algorithm, for example, allows nodes to execute arbitrarily into the future, but must roll back in time whenever they receive events that occurred before their current time [8]. Unfortunately, some events cannot be rolled back, such as rolling back time before an ATM dispensed cash to a customer.

We are motivated by the fact that neither strong or weak event ordering are scalable when the number of nodes increases significantly and the number of non-local events is also significant. In this case, each node must send a message to every other node whenever an event is generated that changes the shared application space. For example, in an online stock market system using strong or weak event ordering, every trade, sell, or purchase would be non-local events which would have to be sent to every participant. Furthermore, these events would need to have a strict ordering (or risk a stock market crash!) and it would be unable to roll back time, due to its real-time nature. Without a scalable event ordering system, an online stock market system would not be viable.

We are interested in N-Trees because they are optimized for event propagation so that the fewest number of nodes are contacted for event ordering. DHTs, on the other hand, are optimized for fast storage and retrieval of data using two functions, *insert(key, value)* and *lookup(key)*. DHTs do provide the kind of mapping we need for the application space – we might map the application space to the key space of a DHT, for example. However, propagating events to neighbors and relocating to new places in the DHT would be too slow for scalable event ordering.

N-Trees have a similar advantage when compared to using multicast over a DHT such as Scribe [9]. N-Trees only require that nodes within the scope of an event exchange messages. With Scribe, one could build multiple multicast trees, depending on the scope of peers in the application space. However, building and maintaining the multicast trees in the face of constant churn, and forcing peers not local to generated events to forward messages would create a large amount of overhead that would affect the scalability of the event ordering mechanism.

III. Definitions

A. Application State Space

The *application state space* is defined as the application state that can be affected by events generated in the system. Any distributed system will typically generate events that other nodes in the system need to know about so that their state can be updated appropriately. Furthermore, these events need to be ordered because some events preclude the occurrence of other events.

For example, in a distributed file versioning system for programming code, the application state space could be defined broadly as all possible file names and function names in the archive. Possible events include file and function modification, creation, and deletion. The state space, in this example, includes two *dimensions*, or variables, which are the file names and function names. Thus, we would say this applications has a two-dimensional state space.

B. Scoped Event

A *scoped event* is an event which is labelled with a scope in the dimensionality of the application state space. The scope is a subspace in the application space that indicates how broad a particular event can affect the state in the application, and therefore how many peers an event must reach.

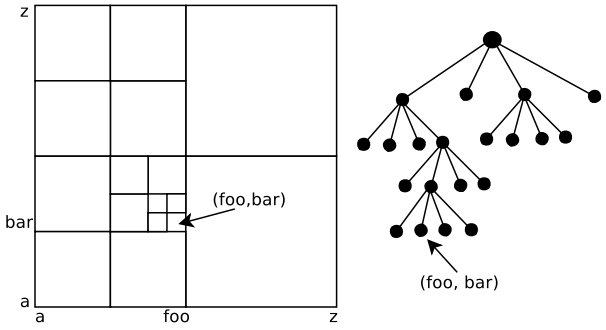

Fig. 1. Cartesian application space and quadtree representation.

Using our previous example, an event that deletes the function *bar* from the file *foo* would be labelled with a scope of *(foo, bar)*. The event would be propagated through the system so that those nodes interested in this event could update their state accordingly.

C. N-Tree

An N-Tree is a generalization of *octree* with n dimensions instead of only three that an octree has, or two that a *quadtree* has [7].

An N-Tree is defined inductively as follows:

- **Base case**: a Leaf
- **Inductive case**: a Node with 2^n children, each an N-Tree

In Figure 1, we illustrate a 2-tree, or quadtree. In a quadtree, the state space is a Cartesian square that is subdivided as necessary to meet the scoping requirements of the application and the current peers in the network. In reference to our example, we label the x-axis with file names and the y-axis with function names (assuming a lexicographic ordering on names). Thus, events generated at *(foo, bar)* in the application state space would occur in the node labelled *(foo, bar)* in the N-Tree from our figure.

The purpose of the N-Tree is to divide the application space and reduce the communication requirements between peers that are operating in different scopes in the application space. The nodes in the tree represent scopes of varying sizes, with the root node representing the entire application. Each successive level below the root represents a further breakdown of the scope by 2^n.

For each node in the N-Tree, at least one peer acts as the leader of the subtree rooted at the node. Leaders have three jobs. First, they help new nodes join the tree by forwarding join messages to other parts of the tree. Second, they propagate events up or down the tree according to the event's scope. Last, they determine when to subdivide the node further or collapse the children into a single node.

Each peer declares a scope of interest in the application state space. When they join the application, they are placed in the appropriate point in the N-Tree as determined by their scope. Even though a peer may be acting as the leader of a node in the N-Tree, she is also located at the leaf in the N-Tree that encompasses her scope. Furthermore, peers may locate themselves in several parts of the tree if they are interested in more than one continuous subspace of the application state space.

Using our example, a peer that is going to modify the function *bar* in file *foo* would join the N-Tree in Figure 1 at the node labelled *(foo, bar)*. She would then learn of other peers interested in modifying that function and would have to resolve the order of events directly with those peers.

The N-Tree is also used for event propagation. When the scope of event that is generated by a peer exceeds the scope of the node it is generated from, the leader of the node propagates the event up the N-Tree. Higher nodes then propagate the event to those children whose scope intersects that of the event. This forwarding process continues until all nodes within the scope of the event are reached.

Because the main purpose of the N-Tree is to reduce communication and propagate events, leaves in the N-Tree are subdivided whenever the population of a leaf exceeds a given threshold. This threshold is application specific and depends on the communication requirements of the application. An interactive application with a large number of peers and high event rate, for example, may have a low threshold so that network links are not overwhelmed by the communication requirements at the leaf. Thus, when the leader of a leaf discovers that the population of the node has exceeded the threshold, the leaf is subdivided into n children and the peers are placed into the appropriate child nodes.

IV. THE N-TREE PROTOCOL

In this section, we describe how we build and maintain an N-Tree while handling joins and leaves. We use a DHT as a bootstrap mechanism to discover nodes in the N-Tree. Nodes use the DHT to register their IP addresses with locations in the application space. When a new node wishes to join the system, they do a lookup on their location in the application space, which returns a list of nodes in that area.

The advantage of the DHT is that if some node in the DHT has failed and entries are missing, the next

responsible node in the DHT will respond with the entries it has. Thus, as long as a single node in the N-Tree can be discovered on the DHT, new peers can join the system.

A. Joining the N-Tree

To join, a peer queries the DHT for some set of nodes in her location of the application space. She then sends a *join* message to one of the nodes that includes the peer's current scope in the application space. The receiving node looks at the scope of the join and determines whether to forward the message up or down the tree. Eventually, the *join* message reaches a node that is within the peer's event scope, and this node notifies the peer so that it can join the N-Tree at this location.

Each node further has the ability to divide its scope based on the communication needs of the application. The reason that subdivision is application specific is that some applications have a much higher event rate than others. Applications with a high event rate will probably need to subdivide the state space as much as possible so that the m-way communication between the m peers in a node is reduced as much as possible without hurting the performance (or security) of the application.

When a leaf node discovers that it contains more than a given threshold for the application, it sends a *divide* message to the nodes. The application space is then subdivided evenly along each of the n dimensions. Each node within the newly subdivided space then determines who will act as the leader node (using any appropriate leader election protocol [10]), and the leaders notify their parent of their leadership status. In this manner, the tree can be subdivided as necessary.

B. Leaving the N-Tree

To leave the N-Tree gracefully, a peer sends a *leave* message to its parent node and its group, with a new leader being elected if the leaving peer is the group's current leader. For an ungraceful leave, either a parent or a group will notice that a peer is missing when they try to forward an event to it. The case where the peer is not a leader is trivial. However, when the peer is a leader, the protocol must handle rejoining the group to the tree. If the group noticed that the leader is missing before the parent did, then they can simply re-elect a new leader and notify their parent.

On the other hand, if the tree is attempting to forward an event to a subtree, then it may discover that a child is not responding to an event. To handle this case, leaders should periodically send membership lists to their parent so that the parent can quickly send the event to another group member. Furthermore, the parent can queue events until the subtree is reconnected. However, even if the membership list of the parent is outdated, the parent can locate members of the subtree through the DHT.

If a peer leaves that is both a leader of a group and of one or more parent nodes, then the group will first elect a new leader and then the leader will contact other leaders to elect higher level node leaders. The DHT allows peers to discover other leaders if they do not already know of them.

When peers leave, the leader of the node determines if the subtree can be collapsed. Subtrees are collapsed whenever the total number of peers in the subtree is less than or equal to n in order to prevent unnecessary event propagation up and down the tree.

C. Event Propagation

Event propagation occurs when the scope of an event exceeds the scope of a node in the N-Tree. When a peer generates an event, the event is exchanged with other peers in the same node. When the leader receives the event, she checks the scope and compares it to the scope under her responsibility. If the scope of the event is not completely contained by her node, she forwards the event to her parent. The parent then forwards the event to all children whose scope intersects with the event scope, and additionally forwards it up the tree again.

V. ANALYSIS OF N-TREES

In order to understand N-Trees as a communication structure for scoped events, we provide a simple analysis of their performance in terms of messaging. As with binary trees, most operations on the N-Tree are $O(d)$ where d is the depth of the N-Tree. Note that the depth of the N-Tree is based on the number of *nodes* in the tree, not the number of *peers*, so that $d = O(\lg n)$, where n is the number of nodes in the tree. The number of nodes in the tree is related to the number of peers by an application specific threshold value, t, that indicates how many peers can communicate before the subspace is divided. Typically, $n = p/t$, where p is the number of peers.

Joining is a $O(\lg n)$ operation, where n is the number of nodes in the DHT. This is due to the routing time required by most DHTs. Once a node is found, a peer will most likely be able to join the N-Tree in constant time. However, if the DHT is not up to date, it can take an additional $O(d)$ messages, where d is the maximum depth of the tree.

Unlike a balanced binary tree, N-Trees do not balance themselves to maintain their height to be logarithmic with the number of nodes in the tree. While subdivision reduces peer communication between peers which are generating events in close proximity in the application space, it does not alleviate longer join times.

Leaving, on the other hand, is a simple $O(k)$ operation, where k is the maximum number of members in a leaf (the application threshold before subdivision). When a node leaves, it must contact the other $k-1$ members to notify them that it is leaving.

To analyze the complexity of subdivision, we simply examine leader election protocols. A constant number of messages are required to initiate subdivision. However, leader election using standard protocols can take only one message to as many as $O(k^2)$ messages, where k is the maximum number of peers in a leaf [10].

However, subdivision has a cost in the amount of state that must be stored at each node. For n-dimensions, each node must store 2^n pointers to children. Applications designers should be motivated to reduce the application state space when possible to avoid a state explosion. We believe that most applications will not have a large number of dimensions in the application state space.

N-Tree collapsing is a $O(1)$ operation. Each node is periodically sending membership lists to its parent. When the parent notices that the number of peers in its subtree is less than or equal to n, the parent sends out a message to the peers and collapses the tree.

Event propagation is a $O(d)$ operation, where d is the maximum depth in the tree. In the worse case, a group at the lowest level in the tree generates a global event which must travel to the root of the tree and back to every other group in the application. As with joining, d is roughly $O(\lg n)$, where n is the number of nodes in the network.

VI. AN APPLICATION WITH N-TREES

In this section, we provide an extensive example to demonstrate how N-Trees and event scoping can be used in a peer-to-peer application. Without a loss of generality, we have chosen peer-to-peer educational simulations (which we simply call *the simulation* henceforth for brevity) to demonstrate the need for N-Trees and event scoping. An example of this application would be simulating the international space station using a 3D virtual environment. Since most students will not have the chance to actually visit the space station, students from around the world could log into the simulation, interact with each other, operate equipment and run experiments for a first hand experience of science on the space station.

We chose this application because its events can be scoped, it generates a large number of events frequently (due to its interactive nature), and it needs scalable event ordering so that it can cope with a large number of students. In essence, P2P interactive simulations push the requirements envelope for event ordering with their interactive, real-time nature. However, one should note that this example extends to the many other distributed applications that require event ordering.

A. Application State Space

The first step in designing a peer-to-peer application for N-Trees is to determine the application space. At first, the application space seems enormous in the number of variables that can be altered. However, when choosing the application space, a balance must be struck between fidelity of event scoping and overhead in maintaining a large dimensional N-Tree.

For our application, we reduce the application space to three dimensions, which correspond to the three dimensions of the virtual world. Thus, the application builds a peer-to-peer 3-tree (or octree). We chose an octree because it maps nicely to the three dimensional virtual space where events occur. For example, a student operating a telescope is an event that occurs at some point in the virtual space station, and only those close by need to be notified of the event. On the other hand, a large-scoped event such as moving the space station to a new location in the galaxy, would propagate the event across the leaves in the octree.

B. Event Scopes

The second step is determining *a priori* what scopes events will have. In the simulation, simple actions such as operating a telescope or collaborating with another student on a scientific experiment have a scope that occurs in a single point in the virtual space station. Other actions, such as powering down a section of the space station for repairs has a scope that encompasses that section of the space station, and hence a larger subspace in the application state space.

C. Building N-Trees

In order to build the N-Tree, we need to have a method for electing leaders when a N-Tree is subdivided. Since the simulation generates a large number events, leaders might be chosen based on their bandwidth, since events propagated to and from the group occur through the

leader, or by the node most centrally located to other nodes to reduce latency to all group members.

At the leaf level, each group totally orders events. For interactive applications, where event ordering must be fast and secure, two event ordering protocols can be used: the lockstep protocol [11] or the NEO protocol [12]. While originally designed for games, these protocols generalize to other interactive applications.

D. Primary Issues

The first important issue with the simulation is how the system handles event ordering. Due to the interactive nature of the simulation, event propagation and ordering requires low latency. If the latency increases too much, students will experience difficulty controlling the simulation and interacting with other students. In addition, without a correct ordering of events, the experience perceived by peers will become inconsistent. If we assume that a majority of events occur with a small scope, then the N-Tree organization of peers will ensure that peers are communicating directly with each other the majority of the time. Thus, latency will be as low as possible. The correct ordering of events will occur naturally from the event ordering protocol being used.

One may ask whether we can do better than $O(d)$ for event propagation using another structure. However, without total knowledge of all students in the simulation and their current positions, we believe we can do no better than $O(\lg p)$, where p is the number of peers in the simulation. We can generalize this statement to other applications. Without special knowledge of all other peers, this may be a lower bound on event propagation.

The second important issue in the simulation is how the system handles failures. In a real-time, interactive simulation, students will become frustrated if they have to wait for several seconds each time a leader is lost. Therefore, one may ask whether we can ensure that leader loss does not cause the simulation to *pause*. In particular, as a simulation population grows in size, the probability that a leader will be lost increases.

For this case, we note that each N-Tree node keeps a membership list of the group members of its child nodes. If we assume the majority of events are local, then a lost leader will be detected with a high probability (and repaired) before the simulation is affected. However, to maintain interactiveness, each leaf group could elect two or more leaders. Events would then be forwarded through the additional leaders, with new leaders elected whenever one leaves.

VII. Conclusion and Future Work

N-Trees and scoped events offer a scalable alternative to event ordering in a peer-to-peer network. While not all peer-to-peer applications have events which can be scoped and require scalable event ordering in terms of the number of peers, our work provides a mechanism to make this class of applications scale.

In our example, we used a large scale, interactive simulation to demonstrate how we can use N-Trees and event scopes for scalable event ordering in a peer-to-peer network. The discussion generalizes to all applications with similar requirements (and may be more forgiving in terms of performance than interactive simulations). We believe that a peer-to-peer network cannot do better than $O(\lg p)$ in event propagation, where p is the number of peers, a property we hope to prove in future work.

Our future work is to evaluate the N-Tree protocol in simulation and on the Internet as a mechanism for efficient event ordering. We hope to validate our hypothesis that N-Trees can be used for scalable event ordering.

References

[1] "Gnutella," http://www.gnutella.com.

[2] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Commun. ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[3] M. Dubois, C. Scheurich, and F. A. Briggs, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *Computer*, vol. 21, no. 2, pp. 9–21, 1988.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM*, 2001, pp. 149–160.

[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, "A Scalable Content-Addressable Network," in *ACM SIGCOMM*, 2001, pp. 161–172.

[6] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms*, 2001, pp. 329–350.

[7] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Hughes, *Computer Graphics*. Addison-Wesley, 1996.

[8] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Lang. Syst.*, vol. 7, no. 3, pp. 404–425, 1985.

[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, "SCRIBE: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.

[10] N. A. Lynch, *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., 1996.

[11] N. E. Baughman and B. N. Levine, "Cheat-proof Playout for Centralized and Distributed Online Games," in *INFOCOM*, 2001, pp. 104–113.

[12] C. GauthierDickey, D. Zappala, V. Lo, and J. Marr, "Low Latency and Cheat-proof Event Ordering for Peer-to-Peer Games," in *ACM NOSSDAV*, June 2004.