Decentralized Message Ordering

Cristian Lumezanu

University of Maryland

College Park, MD 20742

lume@cs.umd.edu

advisors:

Neil Spring

Bobby Bhattacharjee

Abstract

We describe a method to order messages in a distributed system without centralized control or vector timestamps. We show that it is *practical*—little state is required; it is *scalable*—the maximum message load is limited by receivers; and it *performs well*—the paths messages traverse to be ordered are not made much longer than necessary. Our insight is that only messages to overlapping groups need to be sequenced, and that when senders subscribe to their groups, a causal order results.

1 Introduction

We describe a method to order messages in a distributed system without centralized control. Our goal is to ensure a consistent ordered delivery of messages to members of the same groups in distributed multi-player games. Messages may represent actions visible in only some locations and recipient groups include all avatars in those locations. Ensuring that different receivers agree on the order of messages is more important than whether that order represents the order of keystrokes.

Our broader goal is to develop protocols that are scalable because they require no centralized servers or state and that are practical by avoiding guarantees that applications do not need. We distribute the task of ordering messages across *sequencing atoms*. Sequencing atoms assign sequence numbers to messages addressed to groups that share receivers. Our approach is scalable because sequencing atoms handle no more messages than the most active receiver in the network—sequencing atoms exist to order the intersections of group memberships, so do not order more messages than receivers. We separate the task of sequencing across as many sequencing atoms as possible for flexibility in distributing load, then rely on placing related atoms on the same or nearby machines (sequencing nodes) to recover performance.

The insight that makes this possible is that the only destinations that can observe ambiguous order are those that subscribe to the same pairs of groups. Only messages to groups with at least two members in common must be ordered. By ordering those messages in the sequencing network and allowing unrelated messages to be ordered by end stations, we remove the requirement of centralized sequencing or long vector timestamps. The sequence numbers provided by sequencing atoms even allow events to be "committed" without ambiguity: receivers can tell when no prior messages are delayed.

For causal ordering, senders must subscribe to the groups to which they send. This requirement is simple and reasonable because receiving sent messages through the system also serves as an end-to-end reliability check. Our ordering is not total: messages to unrelated groups may be delivered in any (perhaps globally inconsistent) order.

We find that causal ordering can be provided with a decentralized network of sequencing atoms. The performance overhead in traversing a sequencing network is reasonable, and the message state and system state required are practical. Our distributed approach enables performance optimizations: placing sequencers close to senders and receivers, trading message processing load against network load by combining sequencing atoms on the same machine, etc. It is well-suited to sequencing groups whose memberships are correlated in space.

This paper is organized as follows. We present related work in the following section. We then describe the goals, assumptions, and procedures of our protocol in Section 3. We use simulations to measure performance in Section 4. We conclude in Section 5.

2 Related Work

The problem of ordered message delivery has been widely studied in distributed systems. Défago *et al.* [9] present an extensive survey, which we summarize here for brevity. Défago *et al.* organize algorithms by the assumptions they make on the underlying system (synchrony model, failure model, communication model, oracle model) and by the objectives they achieve. Here we focus on the ordering mechanisms.

Symmetric approaches are decentralized: each sender determines the order by appending information to all outgoing messages. The appended information reflects a causal order of messages, which may later be transformed into a total order using a predetermined function. Receivers use the attached information to decide whether to deliver or delay a message. Applications can append different types of information; most use timestamps or sequence numbers [19, 23, 20, 10, 3]. Including this information in each message typically requires nodes to keep a view of the messages they have received and sent.

In asymmetric protocols, order is built by a sender, destination, or sequencer. In sender-based protocols [2, 8, 21], the sender can multicast a message only when granted the privilege, i.e., when it holds a token. In sequencer-based approaches, typically one node is elected as a sequencer and is responsible for ordering messages [18, 12, 22]. More than one sequencer can be present, but only one will be active or relevant at a time [6, 24].

To preserve consistency among game states and guarantee fairness among players, networked multi-player games enforce an unambiguous order of events. Typically, a centralized coordinator resolves all conflicts [13, 15, 16, 14]. Although useful in a local area network, as the network grows larger, centralized approaches do not scale well and provide a central point of failure.

Although most work in decentralized ordering algorithms assumes only a single group, a few consider overlapping groups [12, 17, 11, 1]. Our approach is closest to that of Garcia-Molina *et al.* [12]. In the taxonomy of this section, their approach is asymmetric and sequencer-based: they order messages as they deliver them through a tree of subscriber (destination) nodes. A total order of messages results when messages traverse this tree, assuming, among other typical assumptions for fault-tolerant behavior, that message delay is bounded. The graph is arranged so that messages are sequenced by the destination nodes that subscribe to the most groups, and the task of sequencing messages is overlapped with distribution. We separate these tasks to sequencing atoms, which may be placed on any nodes in the network, and to a distribution tree, which may be tailored to perform well despite distant nodes. Our sequencing atoms, instead of sequencing all messages for a destination, sequence only messages for double-overlaps, in which groups share multiple members in common. Although we provide only causal ordering, we expect that our design makes it possible for sequencing atoms to marshal fewer messages and do less work for each message.

3 The Protocol

Our model of an ordered message delivery system consists of three phases: *ingress*, where messages move from senders to the sequencing network, *sequencing*, where messages traverse sequencing atoms while collecting sequence numbers, and *distribution* where packets leave the sequencing network and are sent to destination nodes. We focus on sequencing; existing multicast delivery schemes can support ingress and distribution.

Our goal is to ensure a consistent ordered delivery of messages to members of the same groups. Our key observation is that when messages are sent to groups with overlapping membership, receivers may make inconsistent decisions about the order of those messages. We call groups that have two or more subscribers in common *double overlapped*, and our approach is to provide a sequence number space for each double-overlapped set of groups. These sequence numbers remove the possibility of inconsistent ordering decisions by receivers. By sending messages through sequencing atoms arranged into a sequencing network, the network determines the order of related messages in a decentralized way.

The sequencing graph is arranged so that sequencing atoms instantiated for double-overlapped groups form paths that group messages can follow. A group may have many sequencing atoms because it may have many double-overlaps with other groups. The paths of messages addressed to doubly-overlapped groups intersect at the sequencer associated with the overlap, ensuring that these messages are ordered.

Sequencing atoms are virtual. They need not be placed on different hosts; in fact, placing atoms on the same host may improve performance. A *sequencing node* is a machine that hosts sequencing atoms. We assume that the group membership matrix—which nodes belong to which groups—is globally known; it can be kept in a distributed data store such as a DHT.

3.1 Operation

Each sequencing atom maintains the following state:

- A sequence number for its overlapped groups.
- A group-local sequence number for the groups it acts as ingress node for.
- A forwarding table to direct messages to the next sequencer for each destination group.
- A reverse-path table listing the previous sequencer in the network for each group.
- An output retransmission buffer for each subsequent sequencer.
- A buffer to store received messages from previous sequencers.

Upon receiving a new message from outside the sequencing network, a sequencer assigns it a group-local sequencer number. The message can be forwarded immediately for distribution if its destination group has no double overlaps. Otherwise, if a group has a double overlap sequenced at this sequencer, the current sequence number for the overlap is added.

The message is then placed in the output buffer and transmitted to the next sequencer (if any) in the path for the group. The message can be removed from the buffer when this sequencer receives an acknowledgment from the next hop. If the message is leaving the sequencer network, it will be sent to a delivery tree and on to group members.

This protocol provides two key properties. First, all members of the same group see messages in the same order, which is a causal order if the sender is also part of the group. Second, all destinations can make an immediate decision of whether to deliver or buffer arriving messages.

3.2 Constructing the Sequencing Graph

The sequencing graph must meet two criteria:

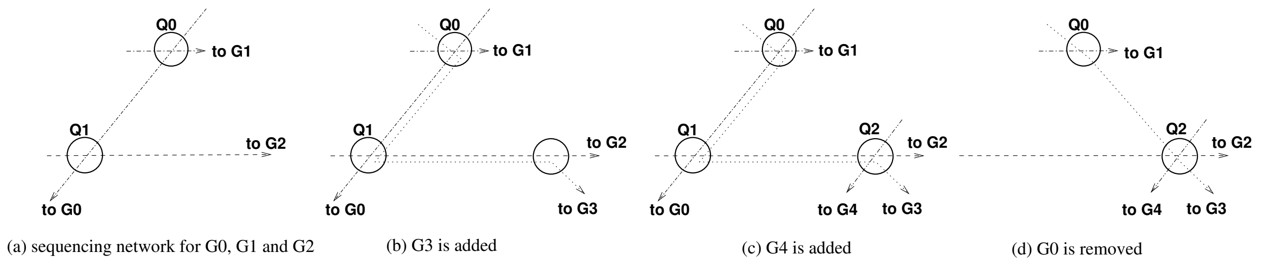

Figure 1: Adding and removing groups: (a) Groups G0 = {A, B, C}, G1 = {B, C}, and G2 = {A, B, D} are served by sequencers Q0 and Q1; (b) G3 = {B, C, D} is added: Q2 will serve the overlap with G2, and Q0 will serve the overlap with G0 and G1; because a direct connection between Q0 and Q2 would create a cycle in the sequencing graph, messages to G3 will be redirected through Q1; (c) G4 = {B, D} is added: messages to G4 will traverse Q2 due to the double overlap among G4, G2 and G3; (d) G0 is removed: sequencer Q1 is no longer needed.

C1: *A single path must connect sequencers associated with each group.*

C2: *The undirected sequencing graph must be loop-free.*

C1 ensures that each message is sequenced relative to all other groups with which the destination group shares a double overlap. When leaving the sequencing network, each message has sufficient information that it can be ordered relative to the messages to all overlapping groups. C2 prevents messages from having circular dependencies, e.g., message a before b, b before c, and c before a. A loop in the sequencing graph could allow an atom to make an ordering decision inconsistent with the ordering of messages not seen by that sequencing atom. The group- and sequencer-based sequence numbers and ordered inter-sequencer message channels ensure a consistent order of related messages at destinations.

Operations on a sequencing network include adding, removing, and modifying groups. We describe only addition and removal; changing the graph when group membership changes can be accomplished by adding a group with the new membership and removing the old one. Figure 1 illustrates these operations;

we describe the figure after presenting the procedure.

Adding the first group G0 is trivial: an ingress-only sequencer is created—this sequencer orders all messages sent to the group. When the second group, G1 is added, if the memberships of G0 and G1 overlap with at least two nodes (are *doubly-overlapped*), a sequencer must represent G0 ∩ G1. All messages for both groups must transit this sequencer, and the G0-specific sequencer may be replaced or removed. This sequencer is *relevant* for all nodes in G0 ∩ G1; the rest need only use the group-local sequence number.

Adding each new group starts with the same basic procedure: a new sequencing atom is instantiated for any new double overlaps. The new sequencing atoms must then be connected to the graph to form a path for the new group so that C1 is satisfied. Unlike C1, C2 is difficult to maintain using only local information. We use a global picture of the sequencing graph and subscription matrix state, stored in a DHT, to find a new sequencer arrangement that satisfies C1 and C2.

Removing a group may eliminate the overlaps that justify a sequencer's existence. Sequencers associated with a group can be removed lazily: adding ignored sequence numbers to a message does not hurt correctness, only efficiency. To remove a group, a termination message is sent to that group, signifying the end of the sequence space for that group, much like a TCP FIN. Each sequencer can inspect this termination message to determine if there is no longer overlap between the nodes this sequencer operates for. If the overlap is gone, the sequencer may retire by informing its parent to forward messages to its child for each sequenced group.

3.3 Placing the Sequencing Atoms

Randomly distributing sequencing atoms throughout the network would lead to poor performance: because messages must traverse the path of sequencing atoms for the group, many needless network hops would result. We have developed heuristics for placing sequencing atoms on nearby nodes and co-locating the sequencing atoms that correspond to related double overlaps on the same node. This arrangement of sequencing atoms on the same sequencing node preserves our scalability goal—that no sequencing machine sees more messages than receivers—without needlessly distributing related sequencing atoms throughout the network.

4 Preliminary Results

In this section, we present a preliminary evaluation of the performance of our ordering scheme.

4.1 Experimental Setup

We developed a packet-level discrete event simulator to evaluate the sequencing protocol. We simulated using a 10,000 node topology generated by GT-ITM [26]. The simulator models the propagation delay between routers, but not packet losses or queuing delays.

We attach hosts to the topology by grouping the hosts into similar size clusters, then distributing each cluster uniformly at random through the topology. Nodes in the same cluster are placed close to each other. We chose this mapping because it is consistent with online gaming communities, in which users tend to cluster around the lowest-latency server.

In our experiments, we vary the number of end-hosts between 32 to 128, and each host can subscribe to zero or more groups. We vary the number of groups from 8 to 32. We generate the size of each group using a Zipf distribution which is known to characterize the popularity of online communities [25, 4], except in one experiment where we vary the density of group membership.

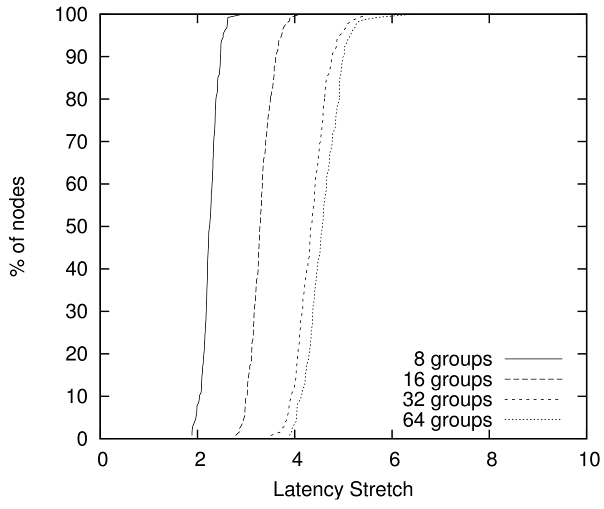

Figure 2: Latency stretch for 128 subscriber nodes, when varying the number of groups.

4.2 Latency Stretch

We evaluate the extra delay messages encounter when traversing the sequencing network compared to taking the shortest unicast path. We measure the *latency stretch*: the ratio between the time taken for a message to traverse the network using the sequencers and the time taken using the direct unicast path. Similar metrics have been described by Chu *et al.* [7] (RDP) and Castro *et al.* [5] (RAD).[1] To measure the latency stretch, each node sends a message to each of the groups it is part of, first using the sequencer network and then directly. We average the results and index them by destination nodes. We leave group membership fixed during the experiment. Our heuristic arranges the sequencing graph and maps it to sequencing nodes—better heuristics may give better results—our intent in this section is to show that acceptable performance is possible.

Figure 2 presents the cumulative distribution of the latency stretch computed for 128 nodes subscribing

[1]RAD is defined per group and RDP per sender-destination pair; we believe latency stretch better represents the performance of our protocol because it captures the delay penalty of an individual node, when the node requires unambiguous delivery.

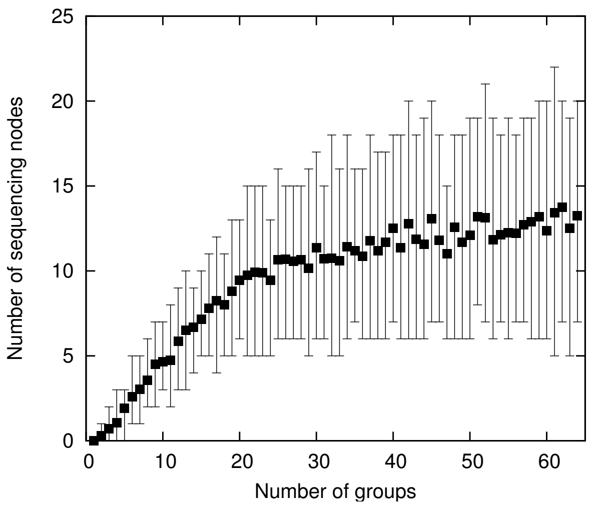

Figure 3: Number of sequencing nodes for 128 subscriber nodes, when varying the number of groups. Error bars indicate 10th and 90th percentiles.

to 8, 16, 32, and 64 groups. When there are fewer groups, the sequencing network is smaller and traversing it takes less time. For example, when we used 8 groups, latency stretch did not exceed 2.5. As the number of groups increases, so do the number of overlaps and the number of sequencing nodes that must be traversed. The growth is sub-linear: for 64 groups, the maximum latency stretch observed was only 5.

4.3 More Groups

We next consider how adding groups affects the number of sequencing nodes. We might worry that the number of sequencing nodes would increase exponentially with the number of groups; such a protocol would be impractical. To simplify presentation, we consider only the sequencing nodes that host non-ingress-only sequencers: each group has at most one ingress-only sequencer, so the ingress-only sequencers may grow linearly with the number of groups.

Figure 3 shows the average number of sequencing nodes created as we vary the number of groups. We vary the number of groups formed by 128 subscriber nodes from 1 to 64, and run the experiment 100 times. The error bars range from 10th to 90th percentile. We observe the same behavior as in Figure 2. As the number of groups increases, there are more overlaps and thus more sequencing nodes. After 20 groups, the number of sequencing nodes grows more gradually. This occurs because many of the new overlaps are identical to, are subsets of, or are supersets of existing overlaps, and so can be mapped to existing sequencing nodes.

4.4 Varied Occupancy

Although we use a Zipf distribution to generate group sizes because we believe it models likely usage, we also wanted to explore worst-case usage scenarios. We define the expected occupancy as a measure of the density of the group membership. The value of the expected occupancy can be interpreted as the probability that a node is member of a group: an occupancy of 0 means that all groups are empty, while an occupancy of 1 means that every node subscribes to every group. Using 128 nodes and 32 groups, we vary the expected occupancy between 0 and 1 to see if the sequencing network approach is more efficient at some group densities.

Figure 4 illustrates how the expected occupancy of groups affects the average number of double overlaps and sequencing nodes. As the expected occupancy increases, the number of double overlaps and necessary sequencing nodes increase until approximately 0.15 occupancy. Beyond this, increasing group densities creates double overlaps that are subsets or supersets of existing overlaps, and the number of sequencing nodes gradually decreases. When the group densities are very high (above 0.9), the overlaps include the entire population and the number of sequencing nodes drops to one.

Although the number of sequencing nodes remains small, the number of overlaps, and thus sequencing atoms, grows large. The size of the graph in atoms is less important, however, than the number of atoms each

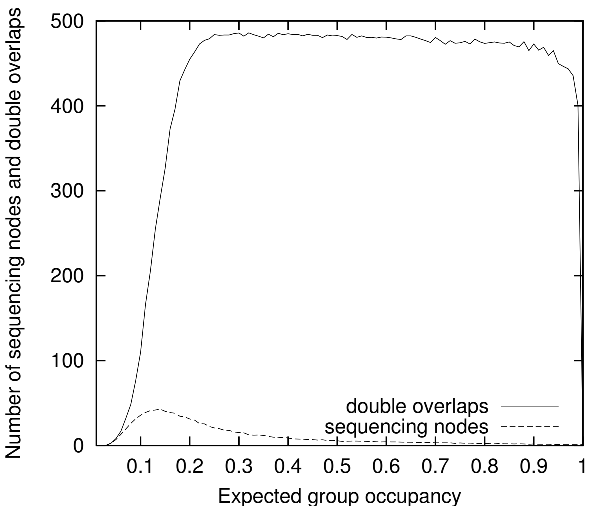

Figure 4: Number of sequencing nodes and double overlaps vs. expected occupancy of groups, for 128 subscriber nodes and 32 groups.

message must traverse, which represents how many sequence numbers a message must collect. We expect that our approach is most attractive when the path length through the sequencing network is smaller than the number of nodes; that is, when the message overhead of sequence numbers provided by the sequencing network is less than that of system-wide vector timestamps. The path length through the sequencing network is bounded by the total number of groups, since a group can have an overlap with at most each of the other groups. As a result, our sequencer-based approach is attractive whenever the number of nodes exceeds the number of groups and when the overlap between groups is small.

5 Conclusion

Our primary contribution is a method for ordering messages in a distributed system without centralized control and without vector timestamps. We showed that it is practical and scalable, because little local and global state is maintained, because sequencing atoms can be placed to achieve good performance relative to a centralized sequencer, and because sequencers order no more messages than destinations receive. Our insight is that only messages to groups with two or more common members must be ordered, and this provides a causal ordering when senders also subscribe.

This approach forms a new primitive for on-line games and publish-subscribe systems. To investigate its applicability, our future work is in applying the idea to the realistic workloads of these and other systems and measuring when group membership is (or can be) geographically-correlated. We also intend to more completely understand the dynamic behavior of our algorithms to determine whether sequencing networks perform well even when incrementally updated as groups and nodes join and leave.

References

[1] M. K. Aguilera and R. E. Strom. Efficient atomic broadcast using deterministic merge. In *PODC*, 2000.

[2] Y. Amir, *et al.* The Totem single-ring ordering and membership protocol. *ACM TOCS*, 13(4):311–342, 1995.

[3] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM TOCS*, 5(1):47–76, 1987.

[4] L. Breslau, *et al.* Web caching and zipf-like distributions: Evidence and implications. In *INFOCOM*, 1999.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal of Selected Areas in Communication*, 2002.

[6] J.-M. Chang and N. F. Maxemchuk. Reliable broadcast protocols. *ACM TOCS*, 2(3):251–273, 1984.

[7] Y. H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, 2000.

[8] F. Cristian. Asynchronous atomic broadcast. In *IBM Technical Disclosure Bulletin*, 1991.

[9] X. Défago, A. Schiper, and P. Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. In *ACM Computing Surveys*, 2004.

[10] D. Dolev, C. Dwork, and L. Stockmeyer. Early delivery totally ordered multicast in asynchronous environments. In *23rd Int'l Symposium on Fault-Tolerant Computing (FTCS-23)*, 1993.

[11] P. D. Ezhilchelvan, R. A. Macedo, and S. K. Shrivastava. Newtop: a fault-tolerant group communication protocol. In *ICDCS*, 1995.

[12] H. Garcia-Molina and A. Spauster. Ordered and reliable multicast communication. *ACM TOCS*, 9(3):242–271, 1991.

[13] L. Gautier and C. Diot. Design and evaluation of mimaze, a multi-player game on the Internet. In *IEEE Int'l Conference on Multimedia Computing and Systems*, 1998.

[14] T. Iimura, H. Hazeyama, and Y. Kadobayashi. Zoned federation of game servers: a peer-to-peer approach to scalable multi-player online games. In *NETGAMES*, 2004.

[15] Y. Ishibashi, S. Tasaka, and Y. Tachibana. A media synchronization scheme with causality control in networked environments. In *IEEE LCN*, 1999.

[16] Y. Ishibashi, S. Tasaka, and Y. Tachibana. Adaptive causality and media synchronization control for networked multimedia applications. In *IEEE ICC*, 2001.

[17] X. Jia. A total ordering multicast protocol using propagation trees. *IEEE Trans. Parallel Distrib. Syst.*, 6(6):617–627, 1995.

[18] M. F. Kaashoek and A. S. Tanenbaum. An evaluation of the Amoeba group communication system. In *ICDCS*, 1996.

[19] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Communications of the ACM*, 1978.

[20] L. L. Peterson, N. C. Buchholz, and R. D. Schlichting. Preserving and using context information in interprocess communication. *ACM TOCS*, 7(3):217–246, 1989.

[21] B. Rajagopalan and P. McKinley. A token-based protocol for reliable, ordered multicast communication. In *8th Symposium on Reliable Distributed Systems (SRDS)*, 1989.

[22] A. Schiper, K. Birman, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM TOCS*, 9(3):272–314, 1991.

[23] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *3rd International Workshop on Distributed Algorithms*, 1989.

[24] B. Whetten, T. Montgomery, and S. M. Kaplan. A high performance totally ordered multicast protocol. In *Selected Papers from the International Workshop on Theory and Practice in Distributed Systems*, 1995.

[25] A. Wolman, *et al.* On the scale and performance of cooperative web proxy caching. In *SOSP*, 1999.

[26] E. W. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. In *INFOCOM*, 1996.