

Epidemic-style Causal Order Broadcasting Only Using Partial View

ChaYoung Kim

Super Computing Center
KISTI
Daejeon, Korea

JinHo Ahn

Dept. of Computer Science
Kyonggi University
Suwonsi Kyonggido, Korea

Abstract - *Because network-level reliable group communication protocols rely on IP multicast and have lack of reliability, this motivates the demand on application-level group communication. Epidemic-style protocols among these application-level approaches guarantee reasonably high reliability, provide good scalability and are easy to deploy. But, earlier versions of these protocols often rely on the assumption that every process knows every other process. There exist epidemic-style broadcast protocols based on partially randomized individual views providing totally ordered delivery of broadcast messages to members of large groups. However, there is no such epidemic-style broadcast protocol providing causally ordered delivery property, which is very useful for many distributed applications such as video-conferencing and multi-party games. This paper proposes an efficient epidemic-style broadcast protocol to guarantee causally ordered delivery semantics based on the local view of every individual member consisting of a subset of members, which continuously evolves, but never exceeds a fixed size.*

Keywords: distributed Systems, group communication, scalability, causally ordered delivery, partial view

1 Introduction

As there are many applications based on a group in distributed systems, the demand of scalable and reliable group communication is very increasing [11]. So, several group communication protocols were proposed based on peer-to-peer interaction models for offering scalability [5]. Techniques such as SRM and RMTP rely on best-effort IP-multicast and ensure reliability by using positive or negative acknowledgments to repair packet losses. But the number of duplicate copies of the multicast that they generate grows linearly with the size of the system. In addition, IP-multicast is not currently deployed in the Internet and tracking membership remains an issue in network-level multicast approaches. Consequently, application-level multicast protocols like SCRIBE [3] and LBRM [6] have recently

received increasing attention. But, LBRM [6] is centralized approaches. It uses loggers to provide stable storage and to handle retransmission of missing messages. So, the amount of information to be stored grows with the number of nodes and loggers could be overloaded. Also, SCRIBE [3] needs the existence of a large-scale peer-to-peer routing infrastructures [5]. In contrast, gossip-based protocols scale well to large groups, are easy to deploy, and degrade system performance gracefully even if node failure or message loss rates increase while they do not need these infrastructures [2,4,9,11].

Gossip-based broadcast protocols appear to be more adequate in large-scale group communication than the traditional strong reliable approaches by Birman [2] and Marzullo [9]. To send a broadcast message, a process randomly selects a subset of members, called gossip-targets. This approach often relies on the assumption that every process knows every other process. When managing large numbers of processes, this assumption becomes a barrier to scalability. It hampers the very nature of a scalable peer-to-peer architecture [4]. So, there have been approaches attempting to address how they scale in terms of membership management [4,5,10,16].

In TIBCO [16], membership management is sometimes delegated to dedicated servers in order to relieve application processes. But this only defers the problem, since those servers are limited in resources. To increase scalability, the membership should be split. In particular, every participating process should have a partial view [4,5]. This partial view of every individual member consists of a subset of members, which continuously evolves and never exceeds a fixed size. Every member selects gossip-targets based on this local view. In order to avoid the isolation of processes, their group membership information should be shared by them to some extent [4]. A protocol that partially addresses this issue is presented in [9,10], where a connection graph called a Harary graph is constructed. However, building such a graph requires the global knowledge of membership, and it may be very difficult to maintain such a graph structure in the presence of joins/leaves of processes. So,

non-deterministic approach is proposed [4,5]. To promote a uniform distribution of membership knowledge among processes, every gossip message also piggybacks a set of process identifiers, which are used to update the views.

But, there is no existing gossip-based causally ordered delivery protocol based on a partial view. Gossip-based protocol offering causally ordered delivery property is very useful for many distributed applications such as video-conferencing, multi-party games and private chat rooms [11]. Therefore, after developing seminal causally ordered broadcast protocols [1,15], further researches have proceeded to minimize the space and the time overheads in dynamically changing broadcast groups [8,12]. Energy efficiency and low bandwidth requirements of the algorithm proposed in [12] are suitable for mobile computing systems. The algorithm presented in [8] achieves optimality by transmitting the bare minimum causal dependency information specified by the necessity conditions, and using an encoding scheme to represent and transmit this information. And in [7], a gossip-based causally ordered delivery broadcast protocol based on a global group view was proposed. In this paper, we present an efficient gossip-based causally ordered delivery protocol based on a partial view consisting of a subset of members, which continuously evolves, but never exceeds a fixed size. This protocol can provide performance improvements in terms of message stability, throughput and membership management compared with the existing causally ordered gossip ones using global views. Moreover, the protocol is not affected by changing the size of large-scale groups and results in very low delivery latency.

2 System Model

The distributed system consists of a finite set of processes $P = \{p_1, \dots, p_n\}$ that communicate only by exchanging messages over a fully connected, point-to-point network. The processes have unique, totally ordered identifiers, and can toss weighted, independent random coins. Runs of the system proceed in a sequence of rounds in which messages sent in the current round are delivered in the next. There are two types of failures, both probabilistic in nature. The first are process failures. There is an independent, per-process probability of at most γ that a process has a crash failure during the finite duration of a protocol. Such processes are called faulty. And processes that never fails are correct. The second type of failures are message omission failures. There is an independent, per-message probability of at most δ that a message between non-faulty processes experiences a send omission failure. The union of all message omission failure events and process failure events are mutually independent. There are no malicious faults, spurious messages, or corruption of messages (i.e., we do

not consider Byzantine failures.). For simplicity, we do not include process recovery in the model. We expect that both γ and δ are small probabilities. Processes communicate using the primitives `send(m)` and `receive(m)`. Communication links are fair-lossy, but correct processes can construct reliable communication links on top of fair-lossy links by periodically retransmitting messages.

3 Related Work

Birman et al. [2] proposed a gossip-style protocol, called bimodal multicast also `pbcast`, thanks to its two phases: a "classic" best-effort multicast such as IP multicast is used for the first rough dissemination of messages. The second phase assures reliability with a certain probability by using a gossip-based retransmission: every participant in the system periodically gossips about a digest of its received messages, and gossip receivers can solicit such messages from the sender if they have not received them previously. `Astrolabe` algorithm [13] is a gossip-based resource location algorithm for the Internet and can be seen as a membership algorithm in that sense. This algorithm enables the reduction of the view of each individual process. Each process has a precise view of its immediate neighbors, while the knowledge becomes less exhaustive at increasing distance. `Astrolabe` however only considers the propagation of membership information and it is not clear how this membership interacts with `pbcast` [2].

Directional Gossip [9] is especially targeted at wide area networks. By taking into account the topology of the network, optimizations are performed. A weight is computed for each neighbor process, representing the connectivity of that given process. The larger the weight of a process, the more possibilities exist thus for it to be infected by other processes. That way, redundant sends are reduced. The algorithm is also based on partial views, in the sense that there is a single gossip server per LAN that acts as a bridge to other LANs. This however leads to a static hierarchy, in which the failure of a gossip server can isolate several processes from the remaining systems. It is possible to implement less resource intensive broadcast algorithms in terms of memory and network bandwidth based on Harary graph [10]. But it is not clear how these algorithms perform in very large scale WANs where membership is dynamic, that is, in an environment where the members can join and leave the system at runtime. It would be a very difficult task to construct the Harary graph each time the membership changes.

Recently, `Scribe` [11] is proposed for a large-scale event notification infrastructure for topic-based publish-subscribe applications as a decentralized membership protocol, which is built on top of a generic peer-to-peer object location and routing substrate overlaid on the Internet. This overlay

substrate is used to create a topic (group) and to build an efficient broadcast tree for the dissemination of events to the topic's subscribers (members). In opposition to overlay substrate, gossip-based protocols are easy to deploy. But gossip-based broadcast protocols based a global view become a bottleneck at an increased scale. So, in Lpbcast [4] and Directional Gossip [9], the algorithms are proposed gossip-based broadcast membership based on a partial view. Each process has a randomly chosen local view of the system. Lpbcast [4] requires no dedicated messages for membership management. And Scamp [5] employs a self-organizing subscription mechanism that automatically provides each node with a partial view of the membership of fixed size on average. So, Lpbcast and Scamp are completely decentralized membership protocols.

The implementation presented in [15] uses vector clocks. Unlike the first version of ISIS [1], each message carries control information consisting of ordered pairs of the type (destination site, vector time). There can be up to $N-1$ such ordered pairs with each message. In [12], the algorithm is especially suitable for mobile computing systems. In order to maintain causal ordering, each message needs to carry information only about its direct predecessor messages with respect to each destination process and most recent mutually concurrent messages delivered to its sender.

4 The Protocol

The proposed Partial-View based Probabilistic Causally ordered BroadCast (PV-PCBCast) protocol is composed of 15 procedures like in figure 1. We assume that the task scheduler is fair, i.e., all procedures get equal chances to execute. Moreover, each line is executed atomically. When process p starts its execution, the membership sequence number and the message sequence number for p is initialized to zeros, and a buffer for gossip messages and a vector delivered are created. Whenever p broadcasts each message, the message sequence number is incremented by one. In procedure SEND-PCBCast, each message msg p broadcasts is piggybacked with order-list, which contains the summary information about its causally related messages. So, this order-list guarantees causally ordered delivery semantics. In procedure DIGEST-RECEIVED-MSG-PARTIAL-ORDER, order-list is updated. And process p sends this message with a summary of its message histories to randomly selected members. In procedures RUN-PERIODICALLY-GOSSIP and SEND-DIGEST, each process periodically gossips about a digest of all messages in its buffer until the maximum number of rounds has been completed. $msg.gossip-count$ is incremented by one per round.

In procedures ADD-MSG-into-BUFFER(msg), RECEIVE-Unreliable-Multicast(msg), RECEIVE-GOSSIP(msg , $msg-digest$), SEND-GOSSIP-with-

DIGEST(msg), COMPARE-MSG, DELIVER-MSG(msg) and DO-Garbage-Collection, when p receives a unreliable multicast or a gossip message msg from q , it compares msg with messages in its own buffer. If p does neither have nor deliver msg yet, p inserts msg into the buffer. It checks the order-list piggybacked on msg and if the condition is satisfied, both the delayed messages that causally precedes it and the message are delivered to the application. For the integrity property, the vector delivered should be updated to systematically prevent processes from receiving previously delivered messages again. Then, p gossips about msg to a randomly chosen members with a digest after incrementing a gossip-count. $bool-delivered$ of delayed messages is checked with FALSE. Also, some messages with TRUE have been remained in the buffer because of periodically gossiping. During the maximum number of rounds, processes periodically gossip about a digest of messages in their buffer. If the maximum number of gossip count has elapsed after the messages were received, they are garbage-collected.

In procedures RECEIVE-GOSSIP(msg , $msg-digest$), RECEIVE-DIGEST($msg-digest$), RECEIVE-SOLICITATION(ID , $Count$) and SOLICIT-Unreceived-MSG($msg-digest$), when p receives a digest of messages from q , it compares the digest information with messages in its own buffer. If there are some messages p didn't received yet, p sends a solicitation for these messages to the gossip information sender q . If this solicitation is received before the maximum number of rounds of these messages being terminated, q retransmits the messages to p . In procedure VIEW-CHANGE, p periodically updates its local member view. A member that joins a virtual synchrony group will receive all the broadcasts and membership changes with a deterministic reliability. Virtual synchrony can be implemented probabilistically by composing the gossip-style dissemination and K-committee Protocol. The K-committee periodically initiates and agrees on membership changes, then assigns it a global sequence number, and hands it off to gossip-based broadcast protocols for dissemination to the group. The periodic view change offers a checkpoint beyond which messages can be garbage-collected at members in procedure DO-Garbage-Collection. The broadcast dissemination and the repair sub-protocol are blocked during view changes by the K-committee, in order to reduce the number of involuntary member droppings.

Figure 2 illustrates an example of the protocol PV-PCBCast, where $group\ view1=(p1, p2, p3, p4)$, $group\ view2=(p3, p4, p5, p6)$, $view\ size = ||4||$, $fan-out = 2$, maximum number of rounds = 3, ordered message list = ($m1, m2$ or $m3, m4$). The global member view is ($p1, p2, p3, p4, p5, p6$). Every participating process only have a partial view of the system. For example, a process $p1$ knows a subset

```

procedure SEND_PCBCast
id = p ; msg_seq = msg_seq+1
delivered[id] = {msg_seq, VIEW_SEQ}
order_list = call procedure DIGEST_RECEIVED_MSG_PARTIAL_ORDER(SND)
msg = (id, msg_seq, current_round, 0, TRUE, order_list, VIEW_SEQ)
buffer = b buffer U {msg} ; Unreliable_Multicast(msg)

procedure ADD_MSG_into_BUFFER(msg)
buffer = buffer U {msg}
call procedure DIGEST_RECEIVED_MSG_PARTIAL_ORDER(RCV)
call procedure COMPARE_MSG
if ( current_round - msg.round ) <= R then call procedure SEND_GOSSIP_with_DIGEST(msg)

procedure RECEIVE_Unreliable_Multicast(msg)
call procedure ADD_MSG_into_BUFFER(msg)

procedure DO_Garbage_Collection
for all msg in buffer do
if (msg.gossip_count >= MAX_GOSSIP_COUNT or msg.VIEW_SEQ < VIEW_SEQ) then
remove msg from buffer

procedure COMPARE_MSG
for all msg in buffer do
delivered_ready = TRUE
for all msg in msg.order_list do
if (there is msg not in delivered[msg.id]) then delivered_ready = FALSE
break
if (delivered_ready = TRUE) then call procedure DELIVER_MSG(msg)

procedure DELIVER_MSG(msg)
deliver msg to the application ; msg.bool_delivered = TRUE
delivered[msg.id] = {msg.msg_seq, msg.VIEW_SEQ}

procedure DIGEST_RECEIVED_MSG_PARTIAL_ORDER
if (RCV) then
for all item in msg.order_list[msg.id] do
if (there is not item in order_list)
order_list = order_list U {msg.id, msg.seq, VIEW_SEQ}
else
order_list = order_list U {msg.id, msg.seq, VIEW_SEQ}

procedure RECEIVE_GOSSIP(msg, msg_digest)
if (there is not msg in buffer and msg.VIEW_SEQ = VIEW_SEQ) then
call procedure ADD_MSG_into_BUFFER(msg)
call procedure SOLICIT_Unreceived_MSG(msg_digest)

procedure RECEIVE_DIGEST(msg_digest)
call procedure SOLICIT_Unreceived_MSG(msg_digest)

procedure RECEIVE_SOLICITATION(ID, Count)
if (there is msg = (ID, Count) in buffer and (current_round - msg.round) < R) then
send Forward_MSG(msg) to q

procedure SOLICIT_Unreceived_MSG(msg_digest t)
for all msg in msg_digest do
if (there is not msg in buffer and msg.VIEW_SEQ = VIEW_SEQ) then
send solicit_Retransmission(msg.id, msg.vt[msg.id]) to q

procedure RUN_PERIODICALLY_GOSSIP_AND_VIEW_CHG
current_round = current_round + 1 ; call procedure SEND_DIGEST
call procedure VIEW_CHANGE

procedure VIEW_CHANGE
stop sending broadcasts ; send PRE_NEW_VIEW gossip_msg ;
receive ACK_PRE_NEW_VIEW ; waiting stability in K_COMMITTEE ;
VIEW_SEQ = VIEW_SEQ+1 ; send NEW_VIEW gossip_msg ;

procedure SEND_GOSSIP_with_DIGEST(msg)
msg.gossip_count = msg.gossip_count + 1
for all msg in buffer do
msg.gossip_count = msg.gossip_count + 1
for each p in processes with probability rate
send Gossip(msg, digest(buffer)) to p
call procedure DO_Garbage_Collection

procedure SEND_DIGEST
for all msg in buffer do
msg.gossip_count = msg.gossip_count + 1
for each p in processes with probability rate
send Gossip(digest(buffer)) to p
call procedure DO_Garbage_Collection

```

Figure 1. Procedures for every process p in PV-PCBCast

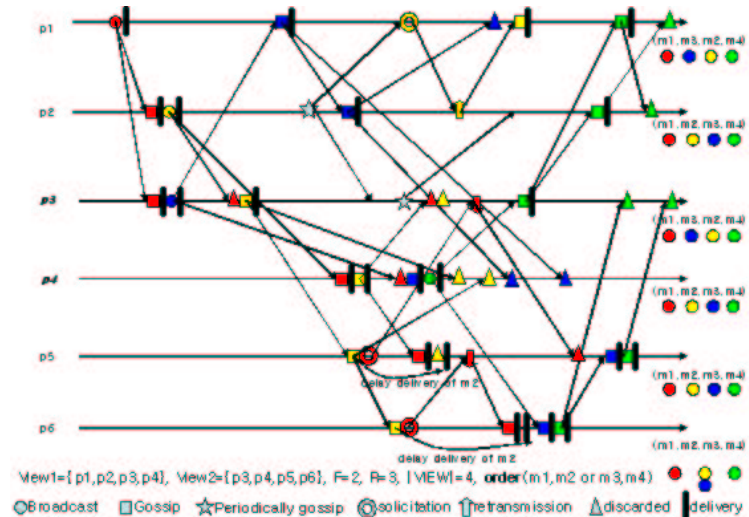


Figure 2. An execution of the proposed protocol PV-PCBCast

of all processes, (p1, p2, p3, p4). The process p1 generates m1 and sends it to randomly selected targets, (p2, p3). After receiving m1, p2 creates m2 and transmits the message, including a gossip message of m1, to (p3, p4). Also, p3 generates m3 and sends the message with m1 to (p1, p4). After receiving m3, p1 immediately delivers it. Then, p1 gossips about m3 to randomly selected processes, (p2, p4). On receiving m1 and m2, p3 knows that they have already delivered m1. So, p3 discards m1 and gossips about m2 to (p4, p5). After receiving m1 and m2, p4 delivers them respectively. Then, it gossips about them to (p3, p5). We know that a gossip message including m1 and m2 is distributed to the partial member view, (p3, p4, p5, p6). Because there are overlapped members in two partitions, messages are received by all of non-faulty members. In here, it is important to avoid the isolation of each partition of the membership especially in the case of failures. Thus, the membership information should be shared by processes to some extent. For instance, process p4 creates m4 and sends it with the gossip message of m3 to (p3, p6). On receiving a gossip message m2, p5 knows it didn't receive m1. Thus, it delays the delivery of m2 and solicits the gossip sender p3 to retransmit m.

5 Conclusion

In this paper, we proposed a probabilistic broadcast protocol PV-PCBCast to guarantee causally ordered delivery semantics based on the local view of every individual member consisting of a subset of members, which continuously

evolves, but never exceeds a fixed size. In the protocol PV-PCBCast, processes propagate each message with the identification information of its causally related messages much like the spread of rumor in society for a fixed number of rounds. Upon receipt of these messages, correct processes immediately deliver the corresponding messages to the application layers in such a way that these deliveries respect causality. Also, the members of the local view are piggy-backed on each sent message. Thus, the view is periodically updated by this gossip-style dissemination. The size of the local view containing nearer neighbors is very smaller than that of the global one. The members of the partial view are joined into a virtual synchrony group. This group will receive all the broadcasts and membership changes with a deterministic reliability. Therefore, the proposed protocol PV-PCBCast can preserve high scalability, low message overheads, decentralized membership management and stable average throughput in large group-based multimedia collaborative works compared with the traditional ones.

References

- [1] K. P. Birman and T. A. Joseph, Reliable Communication in the Presence of Failures, *ACM Transactions on Computer Systems*, Vol.5(1), pp.47-76, 1987.
- [2] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, Bimodal Multicast, *ACM Transactions on Computer Systems*, Vol.17(2), pp.41-88, May 1999.
- [3] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure, *IEEE Journal of Selected Areas in Communications*, Vol.20(8), Oct. 2002.
- [4] P. Eugster, S. Handurukande, R. Guerraoui, A.-M. Kermarrec, and P. Kouznetsov, Lightweight probabilistic broadcast, *ACM Transactions on Computer Systems*, Vol.21(4), pp.341-374, 2003.
- [5] A. J. Ganesh, A.-M. Kermarrec and L. Massoulié, Peer-to-Peer Membership Management for Gossip-Based Protocols, *IEEE Transactions on Computers*, Vol.52, No.2, 2003.
- [6] H. Holbrook, S. Singhal and D. Cheriton, Log-Based Receiver-Reliable Multicast for Distributed Interactive Simulation, *Proc. SIGCOMM*, 1995.
- [7] C. Kim, J. Ahn and C. Hwang, Gossip Based Causal order Broadcast Algorithm, *Proc. ICCSA2004*, 2004.
- [8] A. D. Kshemkalyani and M. Singhal, Necessary and sufficient conditions on information for causal message ordering and their optimal implementation, *Distrib. Comput.* pp.91-111, 1998.
- [9] M.-J. Lin and K. Marzullo, Directional gossip: Gossip in a wide-area network, *Technical Report CS1999-0622*, University of California, San Diego, Computer Science and Engineering, June 1999.
- [10] M.-J. Lin, K. Marzullo and S. Masini, Gossip versus Deterministic Flooding: Low Message Overhead and High-Reliability for Broadcasting on Small Networks, *Proc. 14th Int'l Symp. Distributed Computing*, pp.253-267, 2000.
- [11] D. S. Milojevic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins and Z. Xu, Peer-to-Peer Computing, *Technical Report HPL-2002-57*, HP Laboratories, Palo Alto, March 2002.
- [12] R. Prakash, M. Raynal and M. Singhal, An Adaptive Causal Ordering Algorithm Suited to Mobile Computing Environments, *Journal of Parallel and Distributed Computing*, Vol.41, pp.190-204, 1997.
- [13] R. Renesse, K. Birman and W. Vogels, "Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management, and Data Mining, *ACM Transactions on Computer Systems*, Vol.21, No. 2, pp 164-206, May 2003.
- [14] L. Rodrigues and J. Pereira, Self-Adapting Epidemic Broadcast Algorithms, *Survivability: Obstacles and Solutions 2nd Bertinoro Workshop on Future Directions in Distributed Computing*, June 2004.
- [15] A. Schiper, J. Egli, and A. Sandoz, "A new algorithm to implement causal ordering, *Proceedings of the 3rd International Workshop on Distributed Algorithms*, LNCS, Vol.392, pp.219-232, 1989.
- [16] TICBO. TIB/Rendezvous White Paper. <http://www.rv.tibco.com/>