Scalability Approaches for Causal Broadcasts

R. de Juan-Marín, E. Miedes, H. Decker and F. D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
46022 Valencia - SPAIN

{rjuan,emiedes,hendrik,fmunyoz}@iti.upv.es

Technical Report ITI-SIDI-2009/008

Scalability Approaches for Causal Broadcasts

R. de Juan-Marín, E. Miedes, H. Decker and F. D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
46022 Valencia - SPAIN

Technical Report ITI-SIDI-2009/008

e-mail: {rjuan,emiedes,hendrik,fmunyoz}@iti.upv.es

Abstract

Several current applications of distributed systems demand a highly scalable support, e.g., cloud computing data centres, grid applications, web search engines like Google, Bing, Yahoo!, etc. In most cases such applications replicate their data or their servers. Some kind of reliable broadcast is needed for ensuring the consistency of such replicas, propagating their updates in a pre-determined order (either FIFO, causal or total) at some time. Since total order demands consensus and implies more than a single communication round for delivering each message, causal order seems to be the ideal candidate for scalable broadcasts that ensure an acceptable degree of consistency. This paper surveys different approaches used in the last decades for enhancing the scalability of this kind of communication service.

1 Introduction

Dependable applications are able to ensure high availability and reliability replicating to this end their components. Such replicas should be maintained with some degree of consistency. So, when an updating operation is initiated in one replica, the same operation (or its results) should be propagated to all other replicas and applied there. The degree of replica consistency depends on how the replicas are organised and on the order of update propagation, being the *causal* [13] and *sequential* [19] consistency models two of the most used nowadays, requiring *causal broadcast* [12] and *FIFO atomic broadcast* [12], respectively.

Several modern distributed applications need to provide a highly scalable kind of service. There are multiple examples, being grid applications, cloud computing data centres and web search engines, three of the most important. In order to achieve such degree of scalability, they also need to replicate some of their components. In some cases, there are thousands of nodes collaborating in such applications. As a result of this, a scalable broadcast protocol is needed in these environments. Whilst atomic broadcast is able to provide sequential consistency, it demands consensus in order to decide the message delivery order, needing more than a single communication round in such a process. So, its scalability is compromised. On the other hand, reliable causal broadcast ensures causal consistency, and its protocols do only demand a single communication step in the regular case. So, it seems to facilitate a good basis for scalable communication. Additionally, the first *group communication systems*, like Isis [7], recommended such kind of broadcast and consistency in order to achieve good performance levels and an acceptable programming model.

We assume some knowledge about common algorithms that implement causal broadcast. Such knowledge can be acquainted in the literature, e.g. [6, 26]. Our aim is to explain different approaches known from various papers [1, 2, 3, 11, 14, 15, 16, 17, 20, 21, 23, 24, 25, 26, 27, 29, 31, 32] that deal with enhancing the scalability of causal broadcast services. To this end, Section 2 provides a historical review of the related literature, whilst Section 3 summarizes the main characteristics of the proposed approaches. Finally, Section 4 concludes with an outlook on contemporary and future applications.

2 Historical Review

Let us discuss on the different solutions that have arisen to causal broadcast scalability along time, without entering into technical details, that will be presented in Section 3.

2.1 Inclusion of Causally-Precedent Messages

The *happens-before* relation was proposed by Lamport in [18] as a relevant event ordering mechanism in distributed systems. Instead of real-time timestamping, that becomes useless without a perfect (and impossible) clock synchronisation, *logical clocks* were proposed.

A first generation of causal broadcast protocols was started by the Isis project [5] based on this *happens-before* relation. Such first protocols ensured that no message could be delivered before any of its potential causal predecessors. To this end, an integer label was assigned to each message, following the principles of Lamport's logical clocks. When a message was broadcast, such event also included all its causally previous messages not yet reported as delivered to all their destinations. As a result, each broadcast event could transfer many messages and this did not favour scalability. On the other hand, the Isis system also introduced the *virtual synchrony* model [9] (providing the *group view* concept and ensuring that broadcast messages are always delivered in all their destinations in the same group view) that can be considered the origin of *dynamic group communication* [28] (i.e., the set of members of a group is not static and may vary over time).

Obviously, this first generation of protocols was not efficient nor scalable. Logical clocks were devised for processes using point-to-point communication, and not for process groups using broadcasts. So, logical clocks need to be complemented with other techniques in order to ensure causal delivery in this environment, and piggybacking all non-fully-delivered precedent messages was one of the solutions. However, when the sending rate is high this introduces serious problems since the amount of piggybacked messages could also be prohibitively high.

2.2 Using Vector Clocks

Part of this problem was solved when *vector clocks* were introduced by [11, 21]. Tagging messages with them, the receivers were able to know whether there were other causally preceding messages not yet delivered. Thus, such pending messages do not need to be included each time a subsequent message was broadcast, and their receivers will block the delivery of any causally subsequent messages until they are delivered. This highly reduced the size of the messages being transferred, but introduced other problems. Now, the amount of nodes in the group should be known by all its members (this information was already provided by the underlying *group membership service* [9]), and the vector clocks being used should have a slot per group member. Thus, if the amount of processes in the group is very large, the size of the vector clocks will still be very large, compromising scalability. Nevertheless, the broadcast protocols [6, 26] (both inspired in [29], that was intended for point-to-point communication instead of broadcasts) developed with this approach were efficient and scaled better than those of the previous generation. Many current group communication systems still use nowadays causal protocols of this kind. So, we will consider these vector-clock-based protocols as the starting point for studying further scalability approaches.

2.3 General Vector Compaction

The main problem to be overcome by this new generation of protocols was the length of the vector clocks being used. A first solution was already presented in [6]: to compact such clocks. To this end, each sender only added to its new broadcast messages those vector slots (and their positions in the vector) that had changed from its last broadcast message. However, this does not always reduce the size of the resulting vector clock. It depends on the message transmission rate from each sender and the number of nodes in the system. As a result, other compacting approaches were needed.

Like the compacting approach of [6], Singhal and Kshemkalyani [31] proposed a general (indeed, it used the same principle) and optimal compacting solution for dynamic groups, that was later refined, formalised, and proved optimal in [17]. The performance –i.e., degree of compaction– of these solutions

was also evaluated in [8], showing that in all analysed scenarios their solution is able to introduce no space penalty, and that in an optimal scenario it is able to reduce the clock length to only 8% of its original size. In [25] the compacted causal clock information is propagated in (sometimes separate) *light control messages* (LCM), and the performance evaluation shown in such paper also proves that these mechanisms are able to enhance the scalability.

Another compacting solution was described in [23], although it was focused in the particular case of multiple groups that share some of their processes, but using the same core ideas described above. It was able to manage multiple groups using a single vector clock, with as many entries as groups. This was cheaper than the solutions described in [6, 32] but it also needs resynchronisation messages sometimes.

2.4 Location-Aware Compaction

Considering a preliminary version of [31] (its technical report one), Meldal et al. [22] proposed different and improved compacting solutions based on a static system topology; i.e., they proved that when the interconnecting topology is known in advance, vector clocks can be highly compacted. They gave the general rules to accomplish such compaction and show that several topologies are able to generate minimal clock-related information (e.g., a star) whilst others can not reduce it at all (e.g., a ring).

Such rules were used in [1], proposing a hierarchical topology that was able to compact the vector clocks being used, enhancing thus its scalability.

This solution was still improved in [27] with the introduction of the *causal separator* concept. Such causal separators act as communication *gateways*, filtering the messages transmitted among two or more subgroups (named *causal zones* in [27]). With them, the vector clocks being used in one of such subgroups do not need to be known by the nodes belonging to other subgroups; i.e., vector clocks are indeed private for their holding causal zones. As a result, a system consisting of many nodes could be divided into multiple causal zones with several causal separators, reducing in this way the size of the vector clocks being managed by each node, making thus practical the usage of the causal protocols already described in Section 2.2, since the vector-clock overhead could be negligible with an appropriate system division. Finally, this paper also shows that FIFO point-to-point communication is enough for interconnecting the causal separators.

2.5 System Interconnection

The usage of causal separators, as proposed in [27], was later refined in [3]. The latter limited the number of causal separators in a given causal zone to be a single one. This specialised node was named *causal server* in [3]. In order to interconnect the causal servers of multiple groups, a regular causal broadcast protocol should be used. This generates a hierarchical structure that was named *daisy architecture* in [3]. There may be many layers in such hierarchical architecture; i.e., the resulting structure is not limited to two layers.

This was the first example of a new class of scalability approach that was later known as *systems interconnection* [10] in the context of *Distributed Shared Memory* (DSM) systems. In practice, the causal servers of this solution inherit most of the characteristics of the *overlapping groups* management [32, 6] in regular *group communication systems*, but in this case such management can be relaxed since all messages should be delivered to all existing groups and no filtering is needed. The similarity between both scenarios resides in having some processes (the causal servers) that belong to more than one group, but in the interconnection case they are only responsible for forwarding (without filtering) messages between such groups.

The best property of an interconnecting solution is that the interconnected systems may internally use the protocol of their choice providing a given semantics. There should be an interconnecting protocol able to ensure that same semantics, but it does not depend on the protocols being used in each interconnected system. Although the protocols described in [27] were also able to ensure such property, Baldoni et al. [3] were the first that explicitly stated this.

Other papers presenting other system interconnection solutions were published later. A first example is [14] that describes solutions to interconnect causal and total-order broadcast protocols. Causal interconnection protocols only demanded reliable FIFO communication when two systems are interconnected, and

also the avoidance of interconnection cycles when more than two systems are managed [14, Theorem 7] (similarly, Meldal et al. [22] also proved that when the connecting paths are acyclic the amount of clock-related information to be included in broadcast messages can be reduced, and some other papers [23, 32] proved that an acyclic structure is needed). On the other hand, when total-order interconnection is intended, they show that an intrusive interconnection protocol is needed; i.e., the broadcast messages can not be locally delivered in the sending system until the forwarder node has propagated such message and a global total order has been decided. Formal proofs of the impossibility of achieving total-order interconnection in a non-intrusive way were later given in [20, 2].

A second example that illustrates the modularity of an interconnecting approach is the set of papers [15, 16, 24] that uses no protocol inside each interconnected system. To this end, all such papers rely in timestamping all broadcast messages with the local clock of its sender node. This requires a physical clock synchronisation that such systems achieve using GPS modules in such computers. Their authors argue that such level of physical synchronisation will be enough for most applications. In order to implement the interconnection, a regular vector clock-based protocol is used in [15], like it was in [3], whilst the other two papers need either only the original Lamport clocks [24], or such logical clocks combined with the physical timestamp used in the sender system [16].

3 Scaling Approaches

As it has been already outlined in the previous section, current *group communication protocols* [9] use a one-round protocol in order to implement causal broadcast, tagging each message with a *vector clock* [11, 21]. So, we will take such kind of protocols as the basis to explain some technical details about the scaling approaches that were mentioned in such historical outline.

In the regular protocols based on vector clocks, each node maintains a vector clock with as many components as nodes exist in the system. Such clock entries are updated according to the rules described in [6]:

- Each time a message is broadcast, it is tagged with the local vector clock of its sender, once such vector clock has increased its local entry.
- The receiver p_j increases the entry belonging to another process p_i when a new message is locally delivered and was sent by p_i. However, such a delivery is only possible when the entry of the sender node in the message vector clock is one unit greater than the value of this same entry in the receiver's clock, and all other entries are greater or equal in the receiver's clock than in the message clock; i.e., if we assume that there are n nodes in the system, p_i was the sender, p_j the receiver and $VC(m)$ is the vector clock of a message m, the following should be respected:

  $$VC(m)[i] = VC(p_j)[i] + 1$$

  and $\forall k \in \{1..n\} : k \neq i$

  $$VC(m)[k] \leq VC(p_j)[k]$$

  This means that whilst these conditions are not true, the message is held in a buffer and it is not yet delivered. The local clock update in the receiver node consists only in doing the following once the incoming message has been delivered:

  $$VC(p_j)[i] = VC(p_j)[i] + 1$$

The mechanisms shown above for enhancing causal broadcast scalability have been outlined in Sections 2.3 and 2.5, and are detailed and compared below. Both of them have the same aim: to reduce the amount of additional information that should be included in the broadcast messages, making it as independent as possible from the actual amount of nodes that compose the system.

3.1 Compaction Approach

Although several optimisations were proposed later [17, 8], the basic compaction approach as described in [31] consists in the following. Each process p_i should maintain two vector clocks, named $LU(p_i)$ – last update– and $LS(p_i)$ –last sent–, in addition to the single $VC(p_i)$ one explained above. The $LU(p_i)[j]$ holds the value of $VC(p_i)[i]$ when p_i last updated its component j. On the other hand, $LS(p_i)[j]$ maintains the value of $VC(p_i)[i]$ when p_i sent its last message to p_j. Finally, when p_i sends a new message to p_j it only needs to transfer the entries $VC(p_i)[k]$ such that $LS(p_i)[j] < LU(p_i)[k]$; i.e., it only needs to transfer the sequence of pairs $\langle k, VC(p_i)[k]\rangle$ that comply with such condition.

Note that this mechanism was intended for general causal communication, not necessarily broadcasts. So, in case of using only causal broadcasts, the $LS(p_i)$ vector clock will not be needed since all its entries will hold the value $VC(p_i)[i] - 1$. On the other hand, the $LU(p_i)[j]$ entries maintain in this case the value of $VC(p_i)[i]$ when p_j broadcast its last message. This allows a trivial optimisation: instead of maintaining such $LU(p_i)$ vector clock, p_i only needs to recall the value of its $VC(p_i)$ clock when it broadcast its last message (let us name it as $LM(p_i)$), and p_i only needs to transfer those entries $k \neq i$ such that $LM(p_i)[k] < VC(p_i)[k]$. So, this compacting approach is equivalent to the one described in [6] that has been also outlined in Section 2.3.

3.1.1 Advantages

This compaction approach makes sense when the set of nodes that broadcast messages in a given group is localised; i.e., not all group members achieve the same sending rates and many of them hardly send any message, at least for a given time interval. In such scenario, the nodes that broadcast most frequently will be able to compact a lot the vector clocks associated to their messages.

Assuming that n is the number of system nodes, b is the number of bits needed to maintain a sequence number (i.e., the value of a vector clock entry), m is the number of bits needed to maintain a node identifier, and k is the proportion of clock entries that need to be propagated in each broadcast, this technique is convenient when [31]: $k < \frac{n*b}{m+b}$. This expression is easily held in most systems, as shown in [8].

Additionally, this compaction mechanism is able to adequately manage dynamic groups, where the amount of system nodes varies along time. Indeed, although it was initially presented as a technique to improve scalability, its benefits do not heavily depend on the size of vector clocks being assumed, and it is able to provide important memory saves even in systems that consist of a not very high number of nodes (for instance, the performance analysis given in [8] does only consider systems from 16 to 100 nodes, and the greater the amount of nodes, the better the compaction will be).

3.1.2 Inconveniences

In the general case, the usage of this compaction approach will reduce the needed bandwidth. However, this does not come for free since it requires that each node maintains an additional copy of its vector clock. Nevertheless, this is a negligible cost since this same amount of memory is added in every message when this compaction technique is not used.

3.2 Interconnection Approach

In an interconnection [10, 2] scenario we assume that there are multiple systems that already have an internal causal broadcast service. At least one node (i.e., process) in each of such systems is chosen as its *interconnection server*; i.e., it will run an interconnection protocol, ensuring thus that all messages initially broadcast in its local system are eventually delivered in all other interconnected systems, and the messages broadcast in those remote systems are also eventually delivered in the local system. But such *interconnection server* provides a regular application process interface to its local broadcast service; i.e., it should not interfere in nor modify the local broadcast protocol execution.

Causal separators [27] provided the basis needed to implement this kind of interconnection protocols, since they isolate each one of the systems as different causal zones. As a result, the usage of vector clocks is an internal issue in each one of such systems and many systems can be composed using an interconnection protocol. From the point of view of vector clock compaction, as studied in Section 3.1, the

interconnection approach provides the ideal scenario since interconnection servers can completely "forget" the values of the vector clocks being used in each system. They are not needed at all in order to implement such interconnection protocol.

The resulting interconnection protocol does only need a FIFO transmission of the messages [14] when the interconnection servers are paired, or an independent (i.e., not related in any way with the internal protocols being used in the subgroups) causal broadcast protocol being executed by the set of all interconnection servers [3].

Failures can be easily tolerated when a non-recoverable fail-stop model [30] is being assumed. To this end, an interconnection protocol only needs [3] to demand *fully stable* [7] or *uniform* [12] delivery of broadcast messages; i.e., that the message is not considered as already delivered until all its destinations have acknowledged its reception. This implies that all its receivers still maintain such message in their buffers until it is notified as fully stable; i.e., it is not garbage collected. In case of failure of its sender whilst the message was being broadcast (and considering that it could not be reported as fully stable), any of its receivers will be able to rebroadcast it to all its intended receivers. So, no message is lost using this approach. In order to consider a message as fully stable in its sending subsystem, it needs to be reported as fully stable in all other subsystems. This is easily ensured by the causal servers, as detailed in [3].

3.2.1 Advantages

The first advantage of an interconnection solution is that it limits the usage scope of the vector clocks needed for implementing causal delivery. They are only used internally, in each interconnected system. This guarantees that the size of such vector clocks is kept small and it does not depend on the overall size of the complete system, but on the size of each interconnected subsystem. So, this facilitates the scalability of the causal broadcast service.

An interconnection protocol may also allow [14] that a particular sender selects either to multicast a message to its local subsystem or to broadcast it to the entire system.

Usually, each one of the interconnected systems has internal access to a very fast computer network whilst the links used for implementing the interconnection have a limited bandwidth. Compared with a deployment of a single causal broadcast protocol among all nodes, the interconnection approach minimises the number of packets needed for implementing the broadcast service through these links with limited bandwidth. Recall that the interconnection protocol does only require either a FIFO or causal communication among the interconnection servers of each connected system, and this only implies a message sent among them for each broadcast message. Without an interconnection solution, the sender would have sent multiple point-to-point messages, one for each one of the message destinations/receivers, and this could imply that those inter-system links were traversed multiple times for a given broadcast.

3.2.2 Inconveniences

Although the interconnection server introduces only a short forwarding step, some papers (e.g., [25]) criticise that such application-level processing introduces a non-negligible delay that might not be affordable by all kinds of applications. For instance, media stream broadcasting should comply with real-time constraints and demand specific and scalable causal broadcast protocols [4].

3.3 Comparison

Both scalability approaches are able to reduce the amount of causal information included in the broadcast messages. On one hand, the compaction mechanism reduces the number of entries in a given vector clock. On the other hand, the interconnection solutions limit the scope of such vector clocks since they are able to divide the system into several subsystems that do only need local vectors with a few entries. So both solutions are compatible and may be easily combined, improving thus the overall scalability. Recall that the performance evaluation given in [8] showed that the compacting approach (at least for point-to-point communication) was able to provide a 75% of reduction in the vector size even in systems with only 16 nodes, reaching a 90% reduction in 100-node systems.

4 Conclusions

This paper has surveyed several mechanisms that may be used in order to ensure causal delivery in a broadcast service, providing a historical review of such kind of solutions. The first protocols included a set of precedent messages not yet fully delivered in each broadcast message, ensuring thus that the causal order could be enforced in all receivers. This was simplified when vector clocks were introduced, reducing thus the size of the messages being broadcast, but they still needed to hold some amount of causal information (the vector clocks themselves) and such amount depends on the size of the system. In order to improve the scalability of these protocols, some kind of vector clock compacting mechanism is needed. We have found two types of compaction. The first one only transmits the vector entries that have been modified since the last message broadcast by the same sender. The second one uses the interconnection principle in order to manage vector clocks only inside the interconnected systems. Thus, the vector size only depends on the size of each subsystem and not on the global size of the full set of nodes. Both approaches can be easily combined, making possible the broadcast of messages in systems composed by a large amount of processes. This could facilitate the usage of causal consistency in cloud computing, grid, and other current distributed applications intended for large systems.

References

[1] N. Adly and M. Nagi. Maintaining causal order in large scale distributed systems using a logical hierarchy. In *IASTED Intnl. Conf. on Applied Informatics*, pages 214–219, 1995.

[2] A. Álvarez, S. Arévalo, V. Cholvi, A. Fernández, and E. Jiménez. On the interconnection of message passing systems. *Inf. Process. Lett.*, 105(6):249–254, 2008.

[3] R. Baldoni, R. Friedman, and R. van Renesse. The hierarchical daisy architecture for causal delivery. In *17th Intnl. Conf. on Distr. Comp. Sys. (ICDCS)*, pages 570–577, Baltimore, Maryland, USA, May 1997.

[4] A. Benslimane and A. Abouaissa. Dynamical grouping model for distributed real time causal ordering. *Computer Communications*, 25:288–302, 2002.

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

[6] K. P. Birman, A. Schiper, and P. Stephenson. Lightweigt causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.

[7] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE-CS Press, Los Alamitos, CA, 1994.

[8] P. Chandra, P. Gambhire, and A. D. Kshemkalyani. Performance of the optimal causal multicast algorithm: A statistical analysis. *IEEE Trans. Parallel Distrib. Syst.*, 15(1):40–52, 2004.

[9] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[10] A. Fernández, E. Jiménez, and V. Cholvi. On the interconnection of causal memory systems. In *19th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 163–170, Portland, Oregon, USA, July 2000.

[11] C. J. Fidge. Timestamps in message-passing systems that preserve the partial ordering. In *11th Australian Computing Conference*, pages 56–66, Feb. 1988.

[12] V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. ACM Press, 2nd edition, 1993.

[13] P. W. Hutto and M. Ahamad. Slow memory: Weakening consistency to enhance concurrency in distributed shared memories. In *10th Intnl. Conf. on Distributed Computing Systems (ICDCS)*, pages 302–309, Paris, France, May 1990. IEEE-CS Press.

[14] S. Johnson, F. Jahanian, and J. Shah. The inter-group router approach to scalable group composition. In *19th Intnl. Conf. on Distributed Computing Systems (ICDCS)*, pages 4–14, Austin, TX, USA, June 1999.

[15] S. Kawanami, T. Enokido, and M. Takizawa. A group communication protocol for scalable causal ordering. In *AINA (1)*, pages 296–302, Fukuoka, Japan, Mar. 2004.

[16] S. Kawanami, T. Nishimura, T. Enokido, and M. Takizawa. A scalable group communication protocol with global clock. In *19th Intnl. Conf. on Advanced Information Networking and Applications (AINA)*, pages 625–630, Taipei, Taiwan, Mar. 2005.

[17] A. D. Kshemkalyani and M. Singhal. Necessary and sufficient conditions on information for causal message ordering and their optimal implementation. *Distributed Computing*, 11(2):91–111, 1998.

[18] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[19] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

[20] P. Laumay, E. Bruneton, N. de Palma, and S. Krakowiak. Preserving causality in a scalable message-oriented middleware. In *IFIP/ACM Intnl. Conf. on Distributed Systems Platforms (Middleware)*, pages 311–328, Heidelberg, Germany, Nov. 2001.

[21] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. North-Holland, 1989.

[22] S. Meldal, S. Sankar, and J. Vera. Exploiting locality in maintaining potential causality. In *10th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 231–239, Montreal, Quebec, Canada, Aug. 1991.

[23] A. Mostefaoui and M. Raynal. Causal multicast in overlapping groups: Towards a low cost approach. In *4th IEEE Intnl. Workshop on Future Trends of Distributed Computing Systems (FTDCS)*, pages 136–142, Lisbon, Portugal, Sept. 1993.

[24] T. Nishimura, N. Hayashibara, M. Takizawa, and T. Enokido. Causally ordered delivery with global clock in hierarchical group. In *ICPADS (2)*, pages 560–564, Fukuoka, Japan, July 2005.

[25] S. Pomares Hernández, J. Fanchon, K. Drira, and M. Diaz. Causal broadcast protocol for very large group communication systems. In *5th Intnl. Conf. on Principles of Distributed Systems (OPODIS)*, pages 175–188, Manzanillo, Mexico, Dec. 2001.

[26] M. Raynal, A. Schiper, and S. Toueg. The causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett.*, 39(6):343–350, 1991.

[27] L. Rodrigues and P. Veríssimo. Causal separators for large-scale multicast communication. In *15th Intnl. Conf. on Distr. Comp. Sys. (ICDCS)*, pages 83–91, Vancouver, Canada, June 1995.

[28] A. Schiper. Dynamic group communication. *Distributed Computing*, 18(5):359–374, 2006.

[29] A. Schiper, J. Eggli, and A. Sandoz. A new algorithm to implement causal ordering. In *3rd International Workshop on Distributed Algorithms (WDAG)*, pages 219–232, Nice, France, Sept. 1989. Springer.

[30] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant systems. *ACM Trans. on Computer Sys.*, 1(3), Aug. 1983.

[31] M. Singhal and A. D. Kshemkalyani. An efficient implementation of vector clocks. *Inf. Process. Lett.*, 43(1):47–52, 1992.

[32] P. Stephenson. *Fast Ordered Multicasts*. PhD thesis, Dept. of Comp. Sc., Cornell Univ., Ithaca, NY, USA, Feb. 1991.