A Survey of Scalability Approaches for Reliable Causal Broadcasts

R. de Juan[1], H. Decker[1], E. Miedes[1], J. E. Armendáriz[2], F. D. Muñoz[1]

[1]Instituto Universitario Mixto Tecnológico de Informática
Universidad Politécnica de Valencia
46022 Valencia (Spain)

[2]Departamento de Ingeniería Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona (Spain)

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

Technical Report ITI-SIDI-2009/010

A Survey of Scalability Approaches for Reliable Causal Broadcasts

R. de Juan[1], H. Decker[1], E. Miedes[1], J. E. Armendáriz[2], F. D. Muñoz[1]

[1]Instituto Universitario Mixto Tecnológico de Informática
Universidad Politécnica de Valencia
46022 Valencia (Spain)

[2]Departamento de Ingeniería Matemática e Informática
Universidad Pública de Navarra
31006 Pamplona (Spain)

Technical Report ITI-SIDI-2009/010

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

December 17, 2009

Abstract

Several applications of distributed systems need to be highly scalable. For achieving high availability and fault tolerance, such applications typically replicate their data. Due to the dynamics of growth and volatility of the customer markets of such applications, they need to be hosted by highly scalable and adaptive systems. In particular, the scalability of the reliable broadcast mechanisms used for supporting the consistency of replicas is of crucial importance. Reliable broadcasts propagate updates in a predetermined order (e.g., FIFO, total or causal). Since total order needs more communication rounds than causal order, the latter appears to be the preferable candidate for achieving broadcast scalability, although the consistency guarantees based on causal order are weaker than those of total order. This paper provides a historical survey of different approaches used in reliable causal broadcast protocols. In particular, we focus on protocol mechanisms for enhancing both scalability and adaptability.

1 Introduction

A growing number of dependable applications ensure their high availability and reliability by replicating their components. Typical contemporary examples are web search, grid computing and cloud computing. To various extent, the replicas of such applications have to maintain some degree of consistency. So, when an update 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 the particular application, on how the replicas are organised, and on the order of update propagation.

Nowadays, *causal* [36], *sequential* [43] and *linearisable* (or *atomic*) [35] consistency are three of the most frequently employed consistency models. The former two require *causal broadcast* [21] and *FIFO total-order broadcast* [21], respectively, while the latter consistency mode necessitates specialised or more complex protocols (e.g., [7, 22, 63, 68, 82]).

Sometimes, dependable applications should deal with a large number of clients being therefore necessary that they scale properly. Scalability implies that when a system increases its number of replicas (i.e., when it scales out) it is able to serve more clients (i.e., it should also scale up) with the same QoS characteristics. However, in many cases communication is the bottleneck that prevents these systems from scaling

up. Scalability can be seen also as adaptability. In this sense, the application has to scale properly when the number of users increases and it has to free infrastructure resources when the number of users decreases. This adaptability concept is quite important for current trends like cloud computing where applications are run in huge virtualised systems in order to provide efficient solutions.

Thus, a scalable broadcast protocol for propagating updates to all replicas is needed. Whilst total-order broadcast is able to provide sequential consistency, it demands consensus in order to decide the message delivery order. Unfortunately, consensus entails additional communication rounds, which compromises scalability. On the other hand, reliable causal broadcast ensures causal consistency, and its protocols only need a single communication step in the regular case. So, causal broadcast appears to provide an advantageous basis for scalable communication.

At an early stage, broadcasting systems such as Isis [16] recommended causal communication in order to achieve good performance. They coped with the resulting weakenings of causal consistency by adding other synchronisation mechanisms when needed. Later, regular database replication protocols insisted on stronger consistency guarantees and adopted sequential consistency and total-order broadcasts [78]. However, many recent data management applications have a significantly increased need of scalability, while being flexible enough to partially sacrifice consistency [30, 34]. Such a need of consistency relaxation in order to improve the scalability of services have also been recalled in several theoretical papers [17, 32]. So, they advocate for some kind of weak consistency instead of a strong one. Note that the data storage systems being used in cloud computing are able to perfectly deal with large peaks of read-only requests (e.g., with Amazon S3 [53], since that kind of storage service is highly efficient and affordable for static/immutable contents), but might still be overloaded when write-intensive peaks arise (as illustrated by the Asirra application[1] in [26, page 181]). As a result, causal broadcast would regain some interest in scalable and/or dynamic distributed applications, as suggested by several recent works from both theoretical [9, 49] and practical [26] viewpoints.

To the best of our knowledge, there is yet no published survey of the different mechanisms that are used in the numerous reliable causal broadcast protocols existing today. This paper attempts to fill this void, by identifying approaches that are better suited and thus preferable for high scalability needs of applications with dynamic network topologies. To this end, Section 2 specifies what is understood as reliable causal broadcast. Section 3 provides a historical review of the related literature, whilst Section 4 summarises the main characteristics of the proposed approaches. Later, Section 5 describes other scalability approaches that generally do not match the requirements stated in Section 2 but that could foster the development of new scalable protocols able to comply with causal order. Finally, Section 6 sums up comparing the surveyed causal broadcast mechanisms and Section 7 concludes the paper.

2 Reliable Causal Broadcast

We assume the existence of a distributed application that needs causal propagation of messages to all its nodes, where that application components are executed. Such an application is deployed in multiple centres, and each one consists of multiple computers/clusters interconnected by a fast and dedicated LAN. Despite this, the system is still dynamic, since it is composed of a very large amount of computers that might fail and recover, and that could join and leave the system at will, depending on the amount of clients requesting simultaneous service. Due to this, we are talking about *application-level broadcast*, instead of multicast, since messages should be delivered to all these system nodes and we are assuming that these protocols are deployed on top of the transport layer. Due to this implementation at the application layer, on top of TCP in most cases, almost all surveyed papers have assumed *reliable FIFO channels*; i.e., message delivery is guaranteed as long as the receiver is correct, independently of what happens in the sender side. For the sake of concision, we have also followed such assumption in Section 4 in order to describe the protocols being followed in each broadcast class. Note, however, that there have been a few exceptions that have assumed more realistic *quasi-reliable channels* [75, 1, 71]; i.e., both sender and receiver need to be healthy to guarantee message delivery.

[1] Asirra is a CAPTCHA application that authenticates users by asking them whether the contents of a photograph is a cat or a dog. So, it initially needs to read several images from S3, being read-intensive at that step, but it also needs to maintain some session state, and S3 was not appropriate to this end, since the information being stored needs to be resilient and consistent but for a brief interval.

We assume that the broadcast protocols presented in the next sections have been implemented inside a *group communication system* (GCS) [21] that includes a *group membership service* (GMS). Most protocols need the number of processes that participate in the network, or in a group of network nodes, within which messages are to be broadcast. In some cases, concretely in all protocols mentioned in Sections 3.2 and 3.3, knowledge of this number is crucial for correct behaviour. Hopefully, that information is provided by the GMS, and we will use the identifier n to refer to such value. Once a process join or leave occurs, these protocols take note of the membership change and react accordingly by updating their data structures. In some protocols (e.g., [16, 15, 50, 75]), nothing beyond such a reaction is needed for enabling failure resilience upon the leave of a group member. In that case, we shall not discuss anything else regarding fault tolerance. Otherwise, a brief discussion on failure management will be given.

Since a GCS exists, three different events can be distinguished regarding the transmission of a given message:

1. *broadcast(m)*. The message is being broadcast by its sender process (*sender(m)*).
2. *receive(m)*. The message is being received by a GCS module in the node where a process belonging to the group exists. It is temporarily buffered in such GCS module, until causal order is guaranteed.
3. *deliver(m)*. The message is received by its intended target process from its underlying GCS module. Such delivery complies with the intended causal order. Note that message reception (assuming as such the state reached in the previous *receive(m)* event) can not always guarantee causal order. Sometimes this third step will be named also message reception, but it will be clear in that case that the receiver is the actual process, and not its underlying GCS; i.e., no ambiguity will arise.

Note that a node join or failure implies a *view change* event (i.e., a *view* encompasses the set of nodes that constitute the group/system, and such a set is changed each time a node either joins or leaves the group) in a GCS that provides *virtual synchrony* [16]. This property was already proposed in the first generation of broadcast systems described in Section 3.1. The view-change events have important implications on a reliable broadcast specification: what should be understood as a correct process? one that never fails or one that does not fail in the current system view? This was answered by [69], and we use his specifications in this paper. To begin with, a definition of process correctness into a given view v for a group g is needed:

- *v-correct process*: Let us consider some view v, with $p \in v$. Process p is *v-correct* if:
  1. p installs view v, and
  2. p does not crash while its view is v, and
  3. if v is not the last view of some process in v, then exists view v' installed immediately after v by some process in v and $p \in v'$.

and p is *g-correct* if p is *v-correct* in all its views; i.e., p does not fail nor is excluded from g since it joined g.

With this, reliable broadcast is specified using these four properties:

R1 *(Validity)*. If a g-correct process p broadcasts a message m, it eventually delivers m.

R2 *(Agreement)*. If a g-correct process p delivers a message m in view v^p then all v^p-correct processes eventually deliver m.

R3 *(Integrity)*. For any message m, every g-correct process delivers m at most once, and only if m was previously broadcast by $sender(m)$.

R4 *(Same-view delivery)*. If two g-correct processes p and q deliver m in view v^p (for p) and v^q (for q), then $v^p = v^q$.

So, this implies that a reliable broadcast needs to be delivered by all current group living processes (that will also be alive in the next view), and not only by those processes that never fail.

In this paper we are also interested in causal delivery, inspired by the *happens before* relation defined in [42]. Such relation can be summarised as follows.

Let us assume a distributed system consisting of a set P of processes whose execution could be represented by a sequence of events in each process. Quoting [42], the *happens before* relation denoted by "$\rightarrow$" has this definition:

> The relation "$\rightarrow$" on the set of events of a system is the smallest relation satisfying the following three conditions:
>
> 1. If a, b are events in the same process, and a comes before b, then $a \rightarrow b$.
> 2. If a is the sending of a message by one process and b is the receipt of the same message by another process, then $a \rightarrow b$.
> 3. If $a \rightarrow b$ and $b \rightarrow c$ then $a \rightarrow c$.
>
> Two distinct events a and b are said to be *concurrent* if $a \not\rightarrow b$ and $b \not\rightarrow a$.

Note that $\not\rightarrow$ is the negation of $\rightarrow$. Clearly, this definition entails that $\rightarrow$ is anti-reflexive, anti-symmetric, anti-cyclic and transitive.

So, a reliable causal broadcast protocol should comply with all properties defined above for reliable broadcasts, plus the next one:

R5 *(Causal order)*. If the broadcast of a message m causally precedes (i.e., "*happens before*") the broadcast of a message m', then if some g-correct process p delivers both messages in view v^p then no v^p-correct process delivers m' unless it has previously delivered m.

3 Historical Review

Let us discuss some different solutions that have been proposed for enabling the scalability of causal broadcast in the course of time, without entering into technical details. Those will be addressed in Section 4.

3.1 Inclusion of Causally-Precedent Messages

The first generation of causal broadcast protocols was initiated by the Isis project [14], based on the *happens-before* relation. Those 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 (to be described in Section 4.1). Note that 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 the inclusion of precedent messages in the broadcast event was the first technique of this kind. Thus, 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. However, a high sending rate may introduce serious problems, since the amount of added messages in each broadcast could also grow prohibitively high.

So, the next logical step in this evolution seems to be the design of a mechanism that still preserves some notion of the causal history in each broadcast message, but without requiring the inclusion of entire past messages in such sendings. Thus, *Psync*'s *conversations* [56] provided a first solution in this line. It consisted in including only the identifiers of such precedent messages, but not their contents. This was able to reduce the size of the messages being sent but blocking delivery when any of those past and required messages was not yet received. Such solution was generalised with the introduction of *vector clocks*.

In the context of this survey, *blocking delivery* means that a message, once received, is prevented from being delivered until all its causally precedent messages have already been delivered, and such unordered reception is possible in that protocol, introducing additional delays in such delivery step. Additionally,

non-blocking delivery means that each message can be delivered as soon as it has been received; to this end, the broadcast protocol should ensure that all messages are received by their intended targets in the appropriate causal delivery order.

3.2 Using Vector Clocks

The message-size problem was partly solved when *vector clocks* were introduced by [29, 47]. Tagging messages with such clocks, the receivers were able to infer 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 can block the delivery of any causally subsequent messages until they are delivered. This considerably reduced the size of transferred messages, but introduced other problems. The amount of nodes in the group now has to be known by each group member (that information was provided by the underlying *group membership service* [21]), and the used vector clocks need to 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 be correspondingly large, which compromises scalability. Nevertheless, the broadcast protocols developed for the vector-clock-based approach in [15, 62] (both anticipated in [70], though for point-to-point communication instead of broadcasts) turned out to be efficient and scaled much better than those of the previous generation. To this day, most current group communication systems seem to still use causal protocols of this kind (e.g., Transis [5], JGroups [11], Spread [74],...).

Other recent protocols have been also based on this kind of causal history information in order to tag their messages, but they have also addressed other important issues. For instance, in [71] a protocol that assumes quasi-reliable channels and open groups is provided. In *open groups* [25], not only the members of the system group can multicast messages to its members, but any other process can. Most of the surveyed broadcast protocols were intended for *closed groups* where only group members are able to broadcast messages, compelling external clients to forward their messages to any internal node that later broadcasts each message.

3.3 General Vector Compaction

As already indicated, the main problem of this new generation of protocols was the length of the vector clocks being used. A first solution was already presented in [15]: to compact such clocks by incremental changes. To this end, each sender only sets in 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.

Similar in principle to the compacting approach of [15], [73] proposed a general (i.e., it also considers point-to-point communication and this requires clock-matrices instead of vector clocks) and optimal compacting solution for dynamic groups. It was later refined, formalised and proved optimal in [59, 41]. The degree of compaction of these solutions was evaluated in [20]. That evaluation showed that, in all analysed scenarios, their solution can avoid any penalty of additional space requirements, and that, in an optimal scenario, it is able to reduce the vector length to only 8% of its original size. In [57], the compacted causal clock information is propagated in (sometimes separate) *light control messages* (LCM). The performance evaluation exposed in [57] also proves that these mechanisms are able to enhance scalability.

Another compacting solution was described in [50]. Although it focused on the particular case of multiple groups that share some of their processes, it essentially used the same key ideas described above. It was able to manage multiple groups using a single vector clock, with as many entries as groups. This is cheaper than the solutions described in [15, 75], but sometimes it also needs resynchronisation messages.

3.4 System Interconnection

Inspired in a technical report version of [73], [48] proposed different and improved compacting solutions for static system topologies; i.e., they proved that when all interconnections are known in advance, vector clocks can be highly compacted. They specified general rules to accomplish such compaction and showed

that several topologies (e.g., a star) are able to generate minimal clock-related information, whilst others (e.g., a ring) cannot reduce it at all.

Such rules were used in [1], where a hierarchical topology was proposed, compacting the used vector clocks and enhancing thus its scalability.

This solution was further improved in [66] by the introduction of the *causal separator* concept. Causal separators act as communication *gateways*, forwarding the messages transmitted among two or more subgroups, named *causal zones*. Any causal history information (e.g., vector clocks) used in such a causal zone does not need to be known by the nodes belonging to other causal zones. As a result, a system consisting of many nodes could be divided into multiple causal zones with several causal separators. Hence, the size of the vector clocks managed by each node could be reduced, making thus practical the usage of the causal protocols described already in Section 3.2, since the causal history overhead may become negligible with an appropriate system division. Finally, [66] also shows that FIFO point-to-point communication is enough for interconnecting the causal separators.

The usage of causal separators, as proposed in [66], was later refined in [8]. The latter limited the number of causal separators in a given causal zone to be a single one, its *causal server*. In order to interconnect the causal servers of multiple groups, a regular causal broadcast protocol should be used. This generates a hierarchical structure named *daisy architecture* [8]. There may be many layers in such an architecture; i.e., the resulting structure is not limited to two layers.

This was the first example of a new kind of scalability approach that was later known as *systems interconnection* [28] in the context of *Distributed Shared Memory* (DSM) systems. The best property of an interconnecting solution is that the interconnected systems may internally use the protocol of their choice that provides a suitable semantics (reliable causal delivery in the context of this paper, but system interconnection can also be applied to other ordered delivery properties). There should be an interconnecting protocol that is compatible with the same semantics, but it does not depend on the protocols being used in each interconnected system. Although the protocols described in [66] were already able to ensure this property, [8] were the first to explicitly state it.

Other papers presenting other system interconnection solutions were published later. For instance, [38] 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 [38, Theorem 7]. Similarly, [48] also proved that the amount of clock-related information to be included in broadcast messages can be reduced if the connecting paths are acyclic; also [50, 75] proved that acyclicity is needed. On the other hand, when total-order interconnection is intended, [50, 75] show that an intrusive interconnecting protocol is necessary; i.e., a broadcast message can not be locally delivered in the sending system until the forwarder node has propagated the 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 [44, 4].

An alternative kind of interconnecting approach that also illustrates its modularity is described in [39, 40, 52]. It does not use any protocol inside each interconnected system. To this end, it timestamps all broadcast messages with the local clock of its sender node. That requires a physical clock synchronisation, which can be achieved using GPS modules in the host computers. Many authors argue that such level of physical synchronisation is sufficient for most applications, at least when causality is required [58]. In order to implement the interconnection, a regular vector clock-based protocol is used in [39], as in [8], whilst the approaches in the other two papers either work with nothing but the original Lamport clocks [52], or such clocks need to be combined with the physical timestamp used in the sender system [40].

The mentioned modularity enables the protocols to be appropriately tailored and optimised with regard to the particularities of each system in which they are used; e.g., those described in [39] based on GPS-based physical clock synchronisation. Thus, if an intra-system local network has high bandwidth and low delays, then its performance can be very high, providing high message broadcast rates. In the regular case, however, inter-system networks are not able to provide high bandwidth. So, having a single interconnecting server per system may easily become a bottleneck that limits the overall scalability of this approach. In order to overcome this limitation, multiple servers per system could be deployed, thus setting multiple interconnecting channels between each pair of systems. That option has been proposed and analysed in [23], involving a simple subprotocol that ensures an overall FIFO order among all interconnection channels

for each pair of interconnected systems.

4 Reliable Causal Scaling Mechanisms

Most current group communication systems use a one-round protocol in order to implement causal broadcasts, tagging each message with a *vector clock* [29, 47].

At this stage, we could continue with a technical description of the different mechanisms used for improving the broadcast service scalability. However, for completeness, a brief technical description of each generation of solutions presented in the previous section is going to be given now, highlighting the advantages and inconveniences of each of them.

4.1 First Generation

The first generation of reliable causal broadcasts was directly based on the logical clocks proposed by [42]. Such clocks were defined as follows:

> Let clock C_i for each process p_i to be a function which assigns a number $C_i(a)$ to any event a in that process. The entire system of clocks is represented by the function C which assigns to any event b the number $C(b)$ where $C(b) = C_j(b)$ if b is an event in process p_j.

And such definition also requires the following *clock condition*, where E is the set of system events, and $\rightarrow$ represents the *happens before* relation:

$$\forall a, b \in E : a \rightarrow b \Rightarrow C(a) < C(b)\,. \tag{1}$$

The condition (1) can be maintained by adhering to the following two implementation rules:

- IR1 Each process p_i increments C_i between any two successive events.
- IR2 (a) If event a is the sending of a message m by process p_i, then the message m contains a timestamp $T_m = C_i(a)$. (b) Upon delivering a message m, process p_j sets C_j greater than or equal to its present value and greater than T_m.

Note however that logical clocks were intended for point-to-point communication, not for causal broadcast protocols. So, in the mid-eighties, the first broadcast protocols complemented such logical clocks with other constraints in order to ensure causality. To this end, the CBCAST protocol described in [14] requires the following data structures:

- Each process p_i needs to maintain a buffer BUF_{p_i} in order to hold the messages it has received and also the messages it has sent.
- Each message m, besides its regular contents, holds an identifier $ID(m)$ and the set of its intended receivers $REM\_DESTS(m)$.

In order to broadcast a message, this protocol uses the "$\rightarrow$" relation defined by Lamport. CBCAST was designed as a multicast protocol; i.e., the message might be multicast to a subset of the current group membership. This demands the inclusion of additional checks and steps to the resulting protocol. The protocol can be summarised by the following steps:

1. When p_i broadcasts a message m:
   (a) A transfer packet $\langle m_1, m_2, m_3, \ldots \rangle$ is created, including all messages m' in BUF_{p_i} such that $m' \rightarrow m$ and $REM\_DESTS(m') \neq \emptyset$.
   (b) This transfer packet is transmitted to all other group members.
   (c) When the packet has been sent to another process p_j in the previous step, such process p_j is removed from the $REM\_DESTS(m')$ contained in BUF_{p_i} for each m' in the packet.

2. When a process p_j receives packet $\langle m_1, m_2, m_3, \ldots \rangle$, the following is done for each m_i, in increasing order of i:

   (a) If $ID(m_i)$ already exists in BUF_{p_j}, m_i is a duplicate and is ignored.

   (b) If $p_j \in REM\_DESTS(m_i)$, m_i is placed in the delivery queue for p_j, p_j is removed from $REM\_DESTS(m_i)$ and a copy of m_i is placed in BUF_{p_j}.

   (c) Otherwise, m_i is placed in BUF_{p_j}.

Note that this approach demands the inclusion of all not-yet-stable causally-precedent messages in the packet built for broadcasting a new message. Note also that message stability detection is optimistic in this algorithm: it is only based on local delivery or retransmission, but not on a remote reception acknowledgement. Hopefully, such remote reception acknowledgement is finally used in the garbage collection subalgorithm described in [14, Section 4.4]. So, message losses never arise.

4.1.1 Advantages

When regular logical clocks are used, Lamport's *clock condition* (1) can be ensured, but not its reverse; i.e., $C(a) < C(b) \not\Rightarrow a \rightarrow b$. Note that when two events a and b are concurrent, nothing can be said about their logical clock values. So, in some cases (assuming that a and b have been generated by different processes) $C(a) < C(b)$ will arise being a and b two concurrent events. Summarising, it is impossible to find out (using only such logical clocks) whether two messages are concurrent or not.

To solve this issue, CBCAST includes all causally-precedent messages whose reception in each target process has not yet been confirmed. Additionally, all messages included in the transmitted packets are appropriately ordered. So, causal order is ensured in this protocol and such causal delivery never demands that messages were blocked at their delivery step, since when a message is received all its causally precedent messages have also been received and delivered. Note that none of the protocols explained in the following sections are able to maintain this non-blocking behaviour.

4.1.2 Inconveniences

Although the inclusion of precedent messages is able to ensure the protocol correctness and its non-blocking behaviour, there is a high price to pay: the size of the packets being transmitted could be very large when both the message sending rate and the group size are increased. This seriously compromises scalability and explains why this approach was abandoned when vector clocks were introduced in order to track message causality. Additionally, such causally-precedent messages are maintained in a buffer in each intended receiver until they can be garbage-collected. In order to implement such garbage collection an additional round of messages (a message should be sent by each group member) is needed. Hopefully, this extra round can be piggybacked in other new regular broadcasts, but this still demands a non-negligible amount of memory for keeping such buffers in each node.

4.2 Vector Clocks

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 [15]:

- Each time a message m is broadcast by p_i, it is tagged with the local vector clock of its sender (i.e., $VC(m) = VC(p_i)$), once such vector clock has increased its local entry (i.e., $VC(p_i)[i] = VC(p_i)[i] + 1$).
- When m is received by p_j, it is checked whether entry i (the sender's one) in the message vector clock is one unit greater than the value of this same entry in p_j's clock, and all other entries are greater or equal in p_j's clock than in the message clock; i.e., the following should be respected:

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

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

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

Note that this implies that all causally preceding messages already sent by other group members have already been delivered in this receiving node.

This means that the message is buffered and not yet delivered as long as these conditions are not satisfied. When p_j delivers m, it also increases the local entry belonging to p_i (i.e., $VC(p_j)[i] = VC(m)[i]$).

This algorithm explains how vector clocks are used. They are intended for collecting the knowledge about the logical clocks maintained in all system nodes. Thus, each broadcast message is tagged with the knowledge of its sender about the clocks of each system process. Note also that these vector clock slots are only updated when a given process broadcasts a message (it increases its own slot) or when a process delivers a remote broadcast (it increases the slot of its sender). So, they do not hold all process execution events, but only those directly related to broadcast management: the sending events (even the delivery events are not recorded). Moreover, the constraints being set in order to deliver a received message are able to ensure that no causally-precedent message has been missed. If so arises, message delivery is blocked until such precedent messages are received, accepted and delivered.

4.2.1 Advantages

Vector clocks [29, 47] were able to extend the *clock condition* (1) given by [42]. With them, the reverse of such condition can also be guaranteed; i.e., $VC(a) < VC(b) \Rightarrow a \rightarrow b$. But the rules to compare clocks are also more complex. Thus:

$VC(a) \leq VC(b)$ iff $\forall i, i \in \{1..n\} : VC(a)[i] \leq VC(b)[i]$

And:

$VC(a) < VC(b)$ iff $VC(a) \leq VC(b) \wedge \exists i : VC(a)[i] < VC(b)[i]$

Thanks to this, all causal dependencies among broadcast messages can be easily tracked by such vector clocks. This removes the need of including the causally precedent messages in each broadcast. As a result, message sizes are compacted and scalability on size is improved.

4.2.2 Inconveniences

Unfortunately, the non-blocking behaviour of the previous family of protocols has been lost in this case. When a message is dropped or when the routing being used leads to unordered reception, messages are prevented from being delivered until the missed messages are appropriately re-sent, received and delivered. As a result, this kind of protocols do block message delivery in some cases.

Moreover, although message size has been reduced, vector clock size still depends on the amount of processes that compose the system where such messages are broadcast. Note also that this broadcast service is commonly used by highly available applications that will exist for a long time. Thus, the vector clock slots need to hold long integer numbers. If there are a lot of processes, vector clocks will still be very large, and this might again compromise the scalability of this service.

4.3 Compaction Approach

The vector compaction approach tries to minimise the size of the vector clocks being transferred in each broadcast. To this end, only those slots that have changed and are still unknown for the receiving processes are included in the message timestamp. But there are two different mechanisms able to decide which slots need to be included: one for general causal communication [73] that also admits point-to-point sendings, and another specific to broadcast-based communication [15].

Although several optimisations were proposed later [41, 20], the basic point-to-point compaction approach, not intended for broadcasts, as described in [73] consists of the following. Each process p_i should maintain two additional vector clocks, named $LU(p_i)$ –last update– and $LS(p_i)$ –last sent–, besides $VC(p_i)$ as explained above. The $LU(p_i)[j]$ holds the value of $VC(p_i)[i]$ when p_i last updated its component j. In other words, to remember the last message sent by p_i when it delivered the last message from p_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.

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 [15] that has been also outlined in Section 3.3.

4.3.1 Advantages

The described compaction approach makes sense when the set of nodes that broadcast messages in a given group is localised; i.e., not all group members attain the same sending rates and many of them hardly send any message, at least at times. Thus, the nodes that broadcast more 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 [73]: $k < \frac{n*b}{m+b}$. This expression is easily held in most systems, as shown in [20].

Finally, although it was initially presented as a technique to improve scalability, it is able to provide important memory saves even in systems that consist of not a huge number of nodes (for instance, the performance analysis given in [20] does only consider systems with less than 100 nodes).

4.3.2 Inconveniences

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

4.4 Interconnection Approach

In an interconnection scenario [28, 4], we assume that there are multiple systems that already have an internal causal broadcast service. At least one node (i.e., process) in each such system is chosen as its *interconnection server* (IS). The IS runs an interconnection protocol, ensuring that all messages initially broadcast in its local system are eventually delivered in all other interconnected systems. Analogously, the messages broadcast in remote systems are also eventually delivered in the local system. Note that the IS provides a regular application process interface to its local broadcast service; i.e., it should not interfere with the local broadcast protocol execution.

Causal separators [66] provide 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 4.3, the interconnection approach provides an ideal scenario, since interconnection servers can completely "forget" the values of the vector clocks in each system. They are not needed at all in order to implement such an interconnection protocol.

The fact that such causal history (implemented either by vector clocks, as suggested in Section 4.2, or as a set of precedent messages, as addressed in Section 4.1) does not need to be forwarded to other subsystems was proven in [65]. The resulting interconnection protocol only needs a FIFO transmission of the messages [38] when the interconnection servers are paired, or an independent (i.e., not related in any way to the internal protocols used in the subgroups) causal broadcast protocol executed by the set of all interconnection servers [8].

Although the complete proofs can be found in the cited papers, their justification is intuitive. Let us assume that our system S consists in the interconnection of k subsystems S_i ($1 \leq i \leq k$). Let *prec*(m) be the set of messages that causally precede message m. Let $IS(S_i)$ be the interconnection server process of subsystem S_i. Message m only needs to carry as its causal history some subset of *prec*(m), and it is easy to argue that this subset can be empty. Without loss of generality, let us assume that m was initially broadcast in S_1. For each m' in *prec*(m), m' was delivered to $IS(S_1)$ before m, since $m' \rightarrow m$. Thus, $IS(S_1)$ was able to propagate m' to all other S_i ($i \neq 1$) before m was propagated, since IS processes use FIFO channels [38] or a causal interconnection protocol [8] for message propagation. As a result, all messages in *prec*(m) have been transferred and rebroadcast in each subsystem S_i before m is propagated. Since each message in *prec*(m) has been rebroadcast in each subsystem S_i ($i \neq 1$) by the same $IS(S_i)$ process, all of them are causally related in such subsystems (since they have all been sent by the same sender, and causal order implies FIFO order). Thus, they should and will be delivered in their rebroadcast order. This eliminates the need of any causal history in the interconnection protocol.

Failures can be tolerated when a non-recoverable fail-stop model [72] is assumed. To this end, an interconnection protocol needs [8] to demand *safe* [21] delivery of broadcast messages; i.e., although the message can be optimistically delivered once it has been received, it is not discarded until all its destinations have acknowledged its reception. As a consequence, all its receivers still need to buffer such messages until notified as safe; i.e., they are not garbage-collected. In case a sender fails while broadcasting a message that thus could not yet be reported as safe, any of its receivers is still able to rebroadcast it to all intended receivers. So, no message is lost. In order to consider a message as safe by its sending subsystem, it needs to be reported as safe in all other subsystems. This is easily ensured by the interconnection servers, as detailed in [8].

4.4.1 Advantages

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

An interconnection protocol may also allow [38] that a particular sender chooses 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 limited bandwidth. Compared with deploying 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 only requires either a FIFO or a causal communication among the interconnection servers of each connected system, and that only requires a single message to be communicated for each broadcast message. Without an interconnection solution, the sender would have to emit multiple point-to-point messages, one for each receiver, and this could require that those inter-system links (with severely limited bandwidth) were traversed multiple times for a given broadcast. Thus, without interconnection the overall delivery delay would increase, since such low-bandwidth links will be easily saturated.

Finally, the interconnection approach seems to be the ideal candidate for dealing with *dynamic distributed systems* (either when this concept refers to systems whose membership varies with time, but can be known [69], or when it refers to systems whose composition cannot be recorded and whose algorithms might not depend on such membership [31, 2, 6, 51]). Examples of such systems are those mentioned in the introduction: cloud computing data centres, grid applications, web search engines, etc. Note that the

modularity of this approach allows that each interconnected system may use internally any reliable causal broadcast protocol. So, the implementation of such intra-system protocols is not concerned with the full system membership. Additionally, some of the surveyed papers have shown that intra-system protocols may also not depend on its own membership (e.g., the protocols [39] based on GPS modules for achieving a physical clock synchronisation, thus ensuring causal delivery by timestamping messages). As a result, that kind of solutions is able to easily deal with dynamic systems, ensuring thus an acceptable level of scalability for modern applications.

4.4.2 Inconveniences

Although the interconnection server introduces only a short forwarding step, some papers (e.g., [57]) criticise that such application-level processing introduces a delay that might not be affordable by all kinds of applications. For instance, media stream broadcasting should comply with soft real-time constraints and demand specific causal broadcast protocols [79, 10, 12] that tolerate message losses.

If failures arise, the reconfiguration and recovery of the system could need a non-negligible time if any of the interconnection servers has crashed. In such case, another server process should be chosen in that affected subsystem and its identity should be reported to all other subsystems.

5 Other Scalability Approaches

Besides scalability, all causal delivery mechanisms assessed up to this point have also guaranteed reliability; i.e., the message needs to be delivered to all living destinations (*R2 (Agreement)*), no message can be lost and all of them should be delivered in the appropriate order (*R5 (Causal order)*). Moreover, most of those protocols are integrated into a group communication system providing virtual synchrony [16]. In the last decade there have been new kinds of broadcast protocols able to enhance their scalability by relaxing those constraints: *gossiping* [13, 45, 27, 54, 64, 55, 46] and *structured-overlay-based multicast* [37, 81, 61, 18, 19, 46]. Note that they do not strictly comply with reliable causal broadcast as it has been specified in Section 2; thus, they do not perfectly match the aim of this paper. Indeed, they are also intended for dynamic distributed systems although not as those described in [69], but to those where full system membership cannot be recorded. In such context, Tucci-Piergiovanni defines (in Section 2.3 of [77]) *dynamic reliable broadcast* in a relaxed way. To this end, she maintains *R3 (Integrity)*, although specifying it in two different properties *DRB2 (No creation)* and *DRB3 (No duplication)*, whilst relaxing *R2 (Agreement)*, encompassing it into a new *DRB1 (Validity)* property that is specified as:

> DRB1 (Validity). If, at time moment t, a stable node i broadcasts a message m, then m is eventually delivered by each stable node j such that $j \in O(t)$.

where $O(t)$ is the system overlay at time t (i.e., a structured set that comprehends all system nodes), and a *stable node* is a node that remains alive from an instant t onwards. Note that such specification does not require that nodes that eventually leave the system do deliver reliable broadcasts, even if they would remain in the system at time t and still for an interval after t.

Nonetheless, gossip and structured-overlay multicast could be encompassed into a new research goal: to complement them with causal order without compromising scalability. So, for the sake of completeness they are described in the sequel.

5.1 Gossiping

Examples of alternative scalability techniques can be found in collaborative real-time distributed applications that need to broadcast multiple video or audio streams from different sources. Each message carries a fragment of one of the streams and its delivery is supposed to respect a given deadline. If the message is delayed and violates the deadline, it will be discarded when finally received. The Δ-causal protocol [10], inspired by a previous protocol from [79], is able to ensure such property. For timely message delivery, it also demands some optimisations such as message size reduction. To this end, the protocol replaces the

common vector clocks by a list of identifiers of the direct causally-precedent messages needed to ensure causal delivery, using a technique similar to that described in the *conversations* of [56].

This first example opened the door to a new class of broadcast protocols: the *probabilistic*, *epidemic*, or *gossip-based* ones [13, 45, 27, 54, 64, 55]. They share an important characteristic with Δ-causality: messages might be lost, and full delivery to all intended receivers is not mandatory. That relaxes a lot the reliability of such broadcasts, but boosts scalability and throughput stability [13], since the overall performance of the broadcast protocol does no longer depend on the capability of the slowest node, which usually imposes a severe contention in standard reliable protocols. Thus, gossip protocols are based on sending the message to a random subset of processes (some of the sender neighbours). Each such process sends again the message to a randomly selected subset of processes, and so on. Eventually, the message is received by all (or at least, most) system processes.

Note that this kind of broadcast proceeds in several rounds and it does not demand a complete knowledge of the system membership, but only of a subset of directly connected neighbours that will forward the message again. As a result, these protocols deal comfortably with highly dynamic environments where processes join or leave the system very often, as those presented in the introduction.

There are several gossip-based protocol variants. For instance, *bimodal multicast* [13] is based on two different broadcast approaches. In a first attempt, each message is multicast using an unreliable diffusion targeted to all system nodes. Later, each of its receivers uses gossiping. Gossiping is optimised in several ways in order to guarantee throughput stability, since bimodal multicast is used in soft real time distributed applications. To begin with, message retransmissions are structured in rounds, and are supposed to be requested by those receiver nodes that have detected some message losses. Note that the original protocol was intended for FIFO multicasting, which demands a local counter for tagging each message, so that localising missed messages becomes trivial. In order to ensure throughput stability, the amount of data to be retransmitted in a single round is bound, and retransmission follows a most-recent-first policy. Thus, a faulty or saturated process does not compel any healthy process to wait for it. In case of trouble, messages are considered as lost and the pace of the faster processes is adopted by the slowest ones.

A second variant is *directional gossip* [45], specifically tailored for WANs consisting of multiple interconnected LANs. In this variant, each message is gossiped inside each receiving LAN using any local gossiping mechanism. In order to reduce inter-LAN traffic, each LAN has a *gossip server* that forwards received messages to other remote gossip servers using a directional protocol. In such directional gossip, each server assigns some weight to each one of its accessible remote gossip servers that is proportional to the number of paths existing between them. Once a server needs to gossip a message m, it forwards m to all neighbour servers with a weight lower than K. Once that has been done, it forwards m to up to B remote servers in ascending order of weight. Each message also stores the path it has traversed in the broadcast. With this information, each receiver updates the set of paths that interconnect it with the rest of gossip servers. Whenever any path in a node has not been used for a certain while, it is removed from the node's set of paths, and the remote servers' weights are recomputed accordingly. That way, a fast adaptation of this algorithm to dynamic networks is enabled. Note that this strategy has several similarities to the interconnection approach described in Section 3.4, since inter-LAN or inter-system traffic is highly reduced in both cases.

The *lightweight probabilistic broadcast* (*lpbcast*) protocol is another approach, described in [27]. Besides scalability, delivery latency and throughput stability, it also optimises memory consumption. To this end, *lpbcast* imposes bounds on the number of nodes that may be known by each process, forcing them to always work with a reasonably small-sized partial view of the system. In order to avoid membership partitions, each broadcast message also carries (part of) such partial view in its contents, and receivers update randomly their membership sets with such information. Additionally, the number of buffered messages is also limited. Other gossip protocols bound the set of buffered messages and implement some garbage collection mechanism. But in most of them, the purged messages are randomly selected. In *lpbcast*, the selection criterion consists in estimating the actual propagation of each message, thus discarding those messages that have already been received by most processes.

There has been some work on how to ensure complete reliability with gossiping. To this end, [33] propose two different techniques, *hierarchical gossiping* and *adaptive dissemination*. Both need less messages for completing a broadcast than regular gossiping. However, there is still a price to pay for these optimisations: delivery latency is increased, since the number of needed rounds is not reduced, and a

topologically-aware membership scheme is needed in order to ensure reliability. Thus, the partial view known by each node can not be chosen at random, and this might introduce some overhead. Similar solutions are described in [46], where the focus is on both reliability and inter-LAN traffic reduction (as with *directional gossip* and *systems interconnection*).

As we have seen, many papers have studied message propagation through gossip. Each of them proposes one or several optimisations in order to enhance throughput stability, delivery latency or memory consumption. Such optimisations could be merged into a single protocol. That way, scalability can be improved. Unfortunately, gossip protocols commonly do not deal with ordered delivery, although there are a few exceptions. For instance, *bimodal multicast* [13] implements FIFO order. Causally ordered delivery has been attempted in [3], combining gossip-based propagation with the inclusion of causal history information, as in the first Isis algorithm [14]. Thus, this combination maintains a non-blocking delivery, which is mandatory in a gossip-based broadcast, but relies on deterministic view change notifications in order to implement message garbage collection (which is almost impossible to implement in a large-scale system). So, to the best of our knowledge, there has not been any gossip-based protocol able to ensure both the *agreement* and *causal order* properties presented in Section 2.

5.2 Structured-Overlay-based Multicasts

Structured peer-to-peer systems [6] provide an overlay or communication infrastructure with which both routing and a precise (although in most cases, still partial) membership in a large-scale system can be managed. In particular, frequent joins or leaves of processes are easily dealt with. Thus, on that basis, scalable multicast protocols (e.g., Overcast [37], Bayeux [81], CAN Multicast [61], Scribe [18], ...) can be implemented that have better reliable message delivery guarantees than gossiping.

Multicast protocols used in this kind of systems heavily depend on the overlay infrastructure and on the message propagation strategy. According to [19], there are the following alternatives:

- Regarding the overlay infrastructure, all structured systems are able to generate node IDs that determine the location of each node in the overlay. Such an overlay can be organised as follows:
  - *Hypercube routing*. Each participating process plays the role of a node in such a hypercube, which also determines its set of direct neighbours. Besides holding the identifiers of (some of) the direct neighbours, the routing tables maintained in each node also record the addresses of other nodes situated at relative exponential distances. That reduces the size of the routing tables to an amount which is proportional to the logarithm of the system size, and optimises the number of rounds needed for message propagation.

    This kind of overlay has been used in Chord [76], Pastry [67], and Tapestry [80], to name a few systems.
  - *Cartesian hyper-space*. Each node is identified by its coordinates in Cartesian hyper-space, and receives the "ownership" of some part of the hyper-space around its coordinates. The routing table in a node A then consists of the coordinates (and IP addresses) of the nodes placed around the space owned by A. Routing then means to forward messages to the neighbour node with least distance to the intended message receiver.

    Such kind of overlays have been used in CAN [60].

- Regarding the message propagation strategy:
  - *Flooding*. The message is re-sent through some nodes held in the routing tables. That is similar to gossiping, since the number of entries in such routing tables is small, and multiple rounds are needed in order to complete such multicast. Indeed, there are some gossip protocols that can be included here, e.g. [46].
  - *Spanning tree*. Based on the information contained in the routing tables of each system member, a so-called spanning tree is built, for covering the intended group where multicasts should be sent (the full system in case of broadcasts). Later, that tree is used to appropriately propagate the messages being multicast. This strategy minimises (and in some cases even eliminates)

the reception of duplicate messages in the target nodes of a multicast, but introduces a non-negligible effort for building the spanning tree and adapting it in case of node failures.

The combination of these two parameters, each with two possible variants, yields four classes of multicast strategies based on structured overlays: (a) Hypercube routing + flooding propagation (HF), (b) Hypercube routing + spanning tree propagation (HS), (c) Cartesian hyper-space + flooding propagation (CF), (d) Cartesian hyper-space + spanning tree propagation (CS).

Although the resulting algorithms are able to multicast messages to all living group members, there are some problems that endanger performance. On one hand, messages might need an excessive number of propagation rounds in order to reach all of their destinations, introducing a severe delay. In [37, 18, 19], that is measured by a *relative delay penalty* (RDP), which rates its delivery delay against that generated by regular IP multicast protocols [24] used in WANs. On the other hand, a given message might be received multiple times in each target process. That introduces unnecessary network traffic (i.e., *link stress*) and complicates reception management.

A performance evaluation of the four overlay-based multicast strategies is also given in [19], where it is shown that all strategies highly depend on multiple configuration parameters. After selecting an optimal combination of such parameters for each class, the cited evaluation showed that HS is the best choice, giving RDP values below 2 that are between 20% to 50% better than those measured for CS.

When the link stress metric is considered, the use of a spanning tree still seems to be the best option for message propagation. That same optimal combination (HS) for RDP is the best one regarding *average link stress* (ALS) (up to 15% better than using CS), but CS is the best one regarding *maximum link stress* (MLS) (up to 25% lower than using HS).

Note that, similar to gossiping, flooding reached almost 13000 duplicate messages per multicast in a system consisting of 80000 nodes [19], when its best configurations for RDP and link stress were used. Since the usage of a spanning tree strongly reduces the amount of duplicate messages, this illustrates how a structured overlay might surpass the performance achieved by regular gossiping protocols when scalability is considered.

Unfortunately, causal order is still difficult to ensure in the general case. However, some protocols [18] based on spanning-tree propagation use always the same root node for initiating multicasts, independently of its actual sender (i.e., such sender needs a first step to propagate the message to be broadcast to the tree root). With that, it guarantees a causal total order in such multicasts, but at the price of using a single initiator that sequences all such messages, which again might compromise the scalability of the resulting approach.

6 Summary

In order to sum up, Table 1 is provided specifying the most important characteristics of each presented solution. The broadcasting mechanisms shown in that table are: *First Generation* (FG), *Vector Clocks* (VC), *Compaction Approach* (CA), *Interconnection Approach* (IA), *Gossiping* (GO), and *Overlays* (OV). Note that no protocol in the two latter classes (both GO and OV) is currently able to comply with the causal order property (R5) of reliable broadcast protocols, except when a specific topology is used an total order is also ensured. They are included in this summary since they are scalable mechanisms relevant for comparison purposes.

The characteristics being considered in such table rows are:

Membership. Membership management has important implications on the broadcast performance. Firstly, in some cases membership knowledge is required in order to record the causal history of a given message being broadcast. Secondly, a precise and complete membership information requires an appropriate failure detector and consensus on the membership set. This demands a non-negligible effort. So, two different issues are considered regarding membership. To begin with, if such membership knowledge needs to be complete (Com); i.e., it needs to record the identity of all system living nodes, or suffices to be partial (Par). Later, whether such knowledge needs to be precise (Pre), and this requires consensus on the notifications provided to the processes, or it can be relaxed (Rel). Note

that in this latter aspect, common reliable broadcast protocols provide virtual synchrony [16, 21] and this requires precise membership.

Causal History. This characteristic refers to how the solution ensures causal order. The possible values are: inclusion of causal precedent messages (PM), vector clocks (VC) or compacted vector clocks (CC). It is worth noting that some of the approaches (e.g., IA) have a modular design and can use any causal history management (Any). Note also that PM is the unique policy that always prevents receiver processes from blocking when they deal with causal message delivery.

Structure. This highlights whether the solution should have a logical structure of nodes (Yes) or not (No). This characteristic does not have an important implication in the regular performance of the broadcast protocols. However, it might introduce some overhead in case of failures, since such structure needs to be (partially) rebuilt when some node crashes arise.

Advantages. The advantages observed can be the following: non-blocking delivery (NB), or usage of small messages (SM) due to a minimisation of the causal history information being appended to each message.

Drawbacks. Different drawbacks are considered: usage of large messages (LM), need of garbage collection (GC), blocking behaviour (B), or non-causal delivery (NC). Note that the IA alternative is not necessarily blocking, since it can use inside each subsystem any causal history format, even the inclusion of non-stable causally precedent messages (the approach followed in the first protocols generation). Moreover, the interconnecting protocol may either use another reliable causal broadcast protocol (that can again be based on precedent message inclusion) or on paired FIFO communication (that is also non-blocking at the application layer).

	Broadcasting Approaches					
	FG	*VC*	*CA*	*IA*	*GO*	*OV*
Membership	Com Pre	Com Pre	Com Pre	Par Pre	Par Rel	Par Rel
Causal Hist.	PM	VC	CC	Any	–	–
Structure	No	No	No	Yes	No	Yes
Advantages	NB	SM	SM	SM	NB SM	NB SM
Drawbacks	LM GC	B	B	–	GC NC	NC

Table 1: Characteristics summary.

Finally, regarding the adaptability of these approaches to varying workloads, the two latter (GO and OV) provide the best results since they are able to easily accept new nodes in the system (when high peaks of read-only requests are received) and drop existing nodes when the number of client requests is reduced. This adaptability is facilitated by its relaxed and partial membership management. On the other hand, when reliable message delivery and virtual synchrony are needed, the IA is the unique useful option since it does not demand a full membership management (view management is accomplished in each subsystem) and this allows a precise implementation with negligible overhead, since in a subsystem all the communication channels will provide similar delays and bandwidth, using a fast network in most cases. Additionally, workload variations are easily managed, even when a lot of nodes are involved in them (i.e., when many nodes should join or leave the existing system at once). To this end, administrators do only require an appropriate subsystems deployment, adding or removing entire subsystems to the current system state. Note that in case of a subsystem join, the mechanisms needed for implementing the state transference that ensures system consistency are already provided by most modern view-oriented group communication systems.

7 Conclusions

We have historically reviewed and surveyed several mechanisms used for ensuring reliable causal delivery in broadcast services. The first protocols included in each broadcast message its own causal history; i.e., a set of precedent messages not yet fully delivered in all system processes. Thus, the causal order could be enforced in all receivers. That was simplified when vector clocks were introduced, which reduced the size of the broadcast messages. Still, some amount of causal information (the vector clocks themselves) needs to be kept, in proportion of the system's size. In order to improve the scalability of these protocols, some kind of vector clock compaction mechanism may be used. We have found two types of compaction. The first only transmits the vector entries that have been modified since the last message broadcast by the same sender. The second uses the interconnection principle in order to manage vector clocks only inside the interconnected systems. Thus, the vector size only depends on the amount of processes in each subsystem but not on the global size of the full set of nodes. Both approaches can be easily combined, for enabling reliable message broadcasts in systems consisting of a large number of processes, and for facilitating the use of causal consistency in large-scale applications. Some of the revised papers [38] also propose some interconnection variations (basically, to use an intrusive interconnecting protocol that requires a slight patch in the local broadcast protocol) for ensuring total instead of causal order. With such an approach, sequential consistency can be supported.

There are other approaches for ensuring the scalability of broadcast services, like gossiping or overlay-based multicasts. However, such strategies are bound to applications that may relax agreement or ordered delivery because they do not require fully reliable communication. Nonetheless, there currently are some gossip-based protocols that are able to ensure a high degree of reliability at moderate costs (regarding both latency and bandwidth). They could be combined with non-blocking causal delivery strategies (based on carrying only some recent fragment of the causal history within each message), in order to design efficient and scalable causal broadcast algorithms. However, the development of efficient garbage collection mechanisms for pruning causal histories is still an open issue for highly-reliable gossip-based broadcast protocols. On the other hand, overlay-based multicasts allow the implementation of causal total order broadcasts. They build spanning trees on such overlays, with a single root for all broadcasts [18]. That generates a scalable sequencer-based [25] protocol, with moderate management costs and acceptable delays.

To conclude, there is no clear winner among the diverse scalability approaches. The interconnection solution, accompanied by a compaction approach if some of the interconnected systems have a large membership set, is able to minimise causal history information in the broadcast messages, whilst it is still able to provide some support for virtual synchrony, without requiring a full system membership management. That is an excellent basis for highly-available applications, since it simplifies their recovery protocols. On the other hand, when reliability is not a must, both gossiping and overlay-based protocols are favourable, mainly because they are able to guarantee immediate delivery, sacrificing thus a strict causal order compliance, but boosting scalability.

Acknowledgements

This work has been partially supported by EU FEDER and the Spanish MICINN under grant TIN2009-14460-C03, and by EU FEDER and the Spanish MEC under grant TIN2006-14738-C02.

References

[1] Noha Adly and Magdy 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] Marcos Kawazoe Aguilera. A pleasant stroll through the land of infinitely many creatures. *SIGACT News*, 35(2):36–59, 2004.

[3] JinHo Ahn and ChaYoung Kim. Epidemic-style causal order broadcasting only using partial view. In *Intnl. Conf. on Paral. and Dist. Proc. Techn. and Appl. (PDPTA)*, pages 207–211, Las Vegas, Nevada, USA, June 2006. CSREA Press.

[4] A. Álvarez, Sergio Arévalo, Vicent Cholvi, Antonio Fernández, and Ernesto Jiménez. On the interconnection of message passing systems. *Inf. Process. Lett.*, 105(6):249–254, 2008.

[5] Yair Amir, Danny Dolev, Shlomo Kramer, and Dalia Malki. Transis: A communication subsystem for high availability. In *22nd Annual Intnl. Symp. on Fault-Tolerant Comp. (FTCS)*, pages 76–84, Boston, MA, USA, July 1992. IEEE-CS Press.

[6] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004.

[7] Hagit Attiya and Jennifer L. Welch. Sequential consistency versus linearizability. *ACM Trans. Comput. Syst.*, 12(2):91–122, 1994.

[8] Roberto Baldoni, Roy Friedman, and Robbert 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.

[9] Roberto Baldoni, M. Malek, Alessia Milani, and Sara Tucci Piergiovanni. Weakly-persistent causal objects in dynamic distributed systems. In *25th Symp. on Rel. Dist. Sys. (SRDS)*, pages 165–174, Leeds, UK, October 2006. IEEE-CS Press.

[10] Roberto Baldoni, Michel Raynal, Ravi Prakash, and Mukesh Singhal. Broadcast with time and causality constraints for multimedia applications. In *22nd Intnl. Euromicro Conf.*, pages 617–624, Prague, Czech Republic, September 1996. IEEE-CS Press.

[11] Bela Ban. JGroups - a toolkit for reliable multicast communication. Available at: http://www.jgroups.org, 2009.

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

[13] Kenneth P. Birman, Mark Hayden, Öznur Özkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Trans. Comput. Syst.*, 17(2):41–88, 1999.

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

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

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

[17] Eric A. Brewer. Towards robust distributed systems (abstract). In *19th Annual Symp. on Princ. of Dist. Comp. (PODC)*, page 7, Portland, Oregon, USA, July 2000. ACM Press.

[18] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):100–110, October 2002.

[19] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony I. T. Rowstron, Marvin Theimer, Helen J. Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *22nd Annual Joint Conf. of IEEE Comp. and Comm. Societies (INFOCOM)*, San Francisco, CA, USA, April 2003. IEEE-CS Press.

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

[21] Gregory Chockler, Idit Keidar, and Roman Vitenberg. Group communication specifications: a comprehensive study. *ACM Comput. Surv.*, 33(4):427–469, 2001.

[22] Khuzaima Daudjee and Kenneth Salem. Lazy database replication with snapshot isolation. In *32nd Intnl. Conf. on Very Large Data Bases (VLDB)*, pages 715–726, Seoul, Korea, September 2006. ACM Press.

[23] Rubén de Juan-Marín, Vicent Cholvi, Ernesto Jiménez, and Francesc D. Muñoz-Escoí. Parallel interconnection of broadcast systems with multiple FIFO channels. In *11th Intnl. Symp. on Dist. Obj., Middleware and Appl. (DOA)*, volume 5870 of *LNCS*, pages 449–466, Vilamoura, Portugal, November 2009. Springer.

[24] Stephen E. Deering and David R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[25] Xavier Défago, André Schiper, and Péter Urbán. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372–421, 2004.

[26] Jeremy Elson and Jon Howell. Handling flash crowds from your garage. In *USENIX Annual Technical Conference*, pages 171–184, Boston, MA, USA, June 2008. USENIX Association.

[27] Patrick Th. Eugster, Rachid Guerraoui, Sidath B. Handurukande, Petr Kouznetsov, and Anne-Marie Kermarrec. Lightweight probabilistic broadcast. *ACM Trans. Comput. Syst.*, 21(4):341–374, 2003.

[28] Antonio Fernández, Ernesto Jiménez, and Vicent 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.

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

[30] Shel Finkelstein, Rainer Brendle, and Dean Jacobs. Principles for inconsistency. In *4th Biennial Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, January 2009.

[31] Eli Gafni, Michael Merritt, and Gadi Taubenfeld. The concurrency hierarchy, and algorithms for unbounded concurrency. In *20th Annual Symp. on Principles of Dist. Comp. (PODC)*, pages 161–169, Newport, Rhode Island, USA, August 2001. ACM Press.

[32] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

[33] Indranil Gupta, Anne-Marie Kermarrec, and Ayalvadi J. Ganesh. Efficient and adaptive epidemic-style protocols for reliable and scalable multicast. *IEEE Trans. Parallel Distrib. Syst.*, 17(7):593–605, 2006.

[34] Pat Helland and Dave Campbell. Building on quicksand. In *4th Biennial Conf. on Innovative Data Systems Research (CIDR)*, Asilomar, CA, USA, January 2009.

[35] Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

[36] Phillip W. Hutto and Mustaque 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.

[37] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *4th Symp. on Operating System Design and Implementation (OSDI)*, pages 197–212, San Diego, CA, USA, October 2000. USENIX Association.

[38] Scott Johnson, Farnam Jahanian, and Jigney 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.

[39] Satoshi Kawanami, Tomoya Enokido, and Makoto Takizawa. A group communication protocol for scalable causal ordering. In *AINA (1)*, pages 296–302, Fukuoka, Japan, March 2004.

[40] Satoshi Kawanami, Takeshi Nishimura, Tomoya Enokido, and Makoto 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, March 2005.

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

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

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

[44] Philippe Laumay, Eric Bruneton, Noel de Palma, and Sacha Krakowiak. Preserving causality in a scalable message-oriented middleware. In *IFIP/ACM Intnl. Conf. on Distributed Systems Platforms (Middleware)*, pages 311–328, Heidelberg, Germany, November 2001.

[45] Meng-Jang Lin and Keith Marzullo. Directional gossip: Gossip in a wide area network. In *3rd European Dependable Computing Conference*, volume 1667 of *LNCS*, pages 364–379, Prague, Czech Republic, September 1999. Springer.

[46] Miguel Matos, António Sousa, José Pereira, Rui Oliveira, Eric Deliot, and Paul Murray. CLON: Overlay networks and gossip protocols for cloud environments. In *11th Intnl. Symp. on Dist. Obj., Middleware and Appl. (DOA)*, volume 5870 of *LNCS*, pages 549–566, Vilamoura, Portugal, November 2009. Springer.

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

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

[49] Alessia Milani. *Causal Consistency in Static and Dynamic Distributed Systems*. PhD thesis, Università degli Studi di Roma, "La Sapienza", Roma, Italy, 2006.

[50] Achour Mostéfaoui and Michel 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, September 1993.

[51] Achour Mostéfaoui, Michel Raynal, Corentin Travers, Stacy Patterson, Divyakant Agrawal, and Amr El Abbadi. From static distributed systems to dynamic systems. In *24th Symp. on Rel. Dist. Sys. (SRDS)*, pages 109–118, Orlando, FL, USA, October 2005. IEEE-CS Press.

[52] Takeshi Nishimura, Naohiro Hayashibara, Makoto Takizawa, and Tomoya Enokido. Causally ordered delivery with global clock in hierarchical group. In *ICPADS (2)*, pages 560–564, Fukuoka, Japan, July 2005.

[53] Craig Noeldner and Mike Culver. Scalable media hosting in Amazon S3. Downloadable from: http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1073, November 2007. Amazon Web Services Developer Community Article.

[54] José Orlando Pereira, Luís Rodrigues, M. João Monteiro, Rui Carlos Oliveira, and Anne-Marie Kermarrec. NEEM: Network-friendly epidemic multicast. In *22nd Symp. on Rel. Dist. Sys. (SRDS)*, pages 15–24, Florence, Italy, October 2003. IEEE-CS Press.

[55] José Orlando Pereira, Luís Rodrigues, Alexandre S. Pinto, and Rui Carlos Oliveira. Low latency probabilistic broadcast in wide area networks. In *23rd Symp. on Rel. Dist. Sys. (SRDS)*, pages 299–308, Florianopolis, Brazil, October 2004. IEEE-CS Press.

[56] Larry L. Peterson, Nick C. Buchholz, and Richard D. Schlichting. Preserving and using context information in interprocess communication. *ACM Trans. Comput. Syst.*, 7(3):217–246, 1989.

[57] Saúl Pomares Hernández, Jean Fanchon, Khalil Drira, and Michel 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, December 2001.

[58] Ravi Prakash and Roberto Baldoni. Causality and the spatial-temporal ordering in mobile systems. *Mobile Networks and Applications*, 9(5):507–516, 2004.

[59] Ravi Prakash, Michel Raynal, and Mukesh Singhal. An adaptive causal ordering algorithm suited to mobile computing environments. *J. Parallel Distrib. Comput.*, 41(2):190–204, 1997.

[60] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard M. Karp, and Scott Shenker. A scalable content-addressable network. In *ACM SIGCOMM Conf.*, pages 161–172, San Diego, CA, USA, August 2001. ACM Press.

[61] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Application-level multicast using content-addressable networks. In *3rd Intnl. Wshop. on Networked Group Communication (NGC)*, volume 2233 of *LNCS*, pages 14–29, London, UK, November 2001. Springer.

[62] Michel Raynal, André Schiper, and Sam Toueg. The causal ordering abstraction and a simple way to implement it. *Inf. Process. Lett.*, 39(6):343–350, 1991.

[63] Luís Rodrigues, Nuno Carvalho, and Emili Miedes. Supporting linearizable semantics in replicated databases. In *7th IEEE Intnl. Symp. on Networking Computing and Applications (NCA)*, pages 263–266, Cambridge, USA, 2008. IEEE-CS Press.

[64] Luís Rodrigues, Sidath B. Handurukande, José Orlando Pereira, Rachid Guerraoui, and Anne-Marie Kermarrec. Adaptive gossip-based broadcast. In *Intnl. Conf. on Dep. Sys. and Netw. (DSN)*, pages 47–56, San Francisco, CA, USA, June 2003. IEEE-CS Press.

[65] Luís Rodrigues and Paulo Veríssimo. Causal separators and topological timestamping: An approach to support causal multicast in large-scale systems. Technical Report AR-05/95, Instituto de Engenharia de Sistemas e Computadores (INESC), Lisbon, Portugal, 1995. Available at: http://www.navigators.di.fc.ul.pt/docs/abstracts/separators-icdcs.html.

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

[67] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM Intnl. Conf. on Dist. Sys. Platf. (Middleware)*, volume 2218 of *LNCS*, pages 329–350, Heidelberg, Germany, November 2001. Springer.

[68] Raúl Salinas-Monteagudo, Francesc D. Muñoz-Escoí, José Enrique Armendáriz-Iñigo, and José Ramón González de Mendívil. A performance evaluation of g-bound, a consistency protocol supporting multiple isolation levels. In *3rd Intnl. Works. on Rel. in Decentr. Distr. Sys. (RDDS)*, pages 914–923, Monterrey, Mexico, November 2008. Springer.

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

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

[71] Nicolas Schiper and Fernando Pedone. Fast, flexible and highly resilient genuine FIFO and causal multicast algorithms. In *25th ACM Symp. on Applied Comp. (SAC)*, Sierre, Switzerland, March 2010. ACM Press.

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

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

[74] Spread Concepts LLC. The Spread toolkit. Available at: http://www.spread.org, 2009.

[75] Pat Stephenson. *Fast Ordered Multicasts*. PhD thesis, Dept. of Comp. Sc., Cornell Univ., Ithaca, NY, USA, February 1991.

[76] Ion Stoica, Robert Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Conf.*, pages 149–160, San Diego, CA, USA, August 2001. ACM Press.

[77] Sara Tucci-Piergiovanni. *Concurrent Connectivity Maintenance with Infinitely Many Processes*. PhD thesis, University of Rome "La Sapienza", Rome, Italy, November 2005.

[78] Matthias Wiesmann and André Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. on Knowledge and Data Engineering*, 17(4):551–566, April 2005.

[79] Rajendra Yavatkar. MCP: A protocol for coordination and temporal synchronization in multimedia collaborative applications. In *12th Intnl. Conf. on Dist. Comp. Sys. (ICDCS)*, pages 606–613, Yokohama (Japan), June 1992. IEEE-CS Press.

[80] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *Computer Communication Review*, 32(1):81, 2002.

[81] Shelley Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John Kubiatowicz. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *11th Intnl. Wshop. on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, pages 11–20, Port Jefferson, NY, USA, June 2001. ACM Press.

[82] Vaide Zuikevičiūtė and Fernando Pedone. Correctness criteria for database replication: Theoretical and practical aspects. In *10th Intnl. Symp. on Distr. Obj., Middleware and Appl. (DOA)*, pages 639–656, Monterrey, Mexico, November 2008. Springer.