Scalable Atomic Multicast

Luís Rodrigues
DI-FCUL, Lisboa, Portugal
ler@di.fc.ul.pt

Rachid Guerraoui
DI-EPFL, Lausanne, Switzerland
rachid@lsesun3.epfl.ch

André Schiper
DI-EPFL, Lausanne, Switzerland
schiper@lsesun7.epfl.ch

Abstract

We present a new scalable *fault-tolerant algorithm which ensures* total order *delivery of messages sent to multiple groups of processes. Our algorithm is particularly well suited for large scale systems because: (1) any process can multicast a message to one or more groups of processes without being forced to join those groups; (2) inter-group total order is ensured system-wide but, for each individual multicast, the number and size of messages exchanged depends only on the number of addressees; (3) process failure detection does not need to be reliable.*

Our algorithm also exhibits a modular design. It uses two companion protocols, namely a reliable multicast protocol and a consensus protocol, and these protocols are not required to use the same communication channels or to share common variables with the total order protocol. This approach follows a design methodology based on the composition of (encapsulated) micro-protocols.

1. Introduction

Totally ordered multicast is a fundamental abstraction for building distributed reliable applications. In this paper we propose a new scalable algorithm to implement total order multicast, named SCALATOM (SCALable ATOmic Multicast).

1.1. Characteristics of SCALATOM

Contrary to a broadcast that is targeted to the set of all the processes in a system, a *multicast* can be targeted exclusively to a subset of the processes. Total order multicast ensures that the addressees of every message, agree either to deliver or to not deliver the message, and no two processes deliver any two messages in a different order.

In this paper, we assume a system model where multicast are targeted to pre-defined subset of processes called multicast groups. This model corresponds to the practical situations where process groups are used to manage replicated objects (each group represents a logical replica). A fundamental aspect of our approach is that it assumes an *open* model, i.e., senders do not need to join groups they send messages to, and can potentially send messages to any group in the system with global guarantees of inter-group total order. This differs substantially from the *closed* model offering just *local total order* [4, 15, 24][1].

To illustrate the interest of supporting an open model, consider the example ilustrated by Figure 1, where different providers (clients) advertise their products through many different brokers (servers). Providers disseminate updates about availability and prices of their products through multicasts sent to the set of registered (interested) brokers. Different brokers represent different sets of providers. For availability, brokers are replicated, and total order is used to maintain replica consistency. Using an open model, updates can be disseminated in a single totally ordered multicast. In contrast, if a closed model offering local order was used, all brokers would need to belong to the same group. An alternative approach would be to decouple each update in several multicasts, one for each group, but in this case the "all-or-nothing" property of a single multicast is lost: in the event of the sender's failure, some brokers may receive an update and others may not, leading to undesirable inter-broker inconsistencies.

The motivation of our work is to support an open model of multicast communication in a scalable manner. Before presenting our algorithm, we start by clarifying some issues related to scalability issues in total order algorithms.

1.2. On scalability issues for total order multicast

Implementing a total order multicast algorithm with a total order broadcast is straightforward but not scalable. Indeed, a broadcast involves all the processes in the system, *even those that are not concerned with a message*. Guerraoui and Schiper introduced the notion of *participant minimality* to characterize "genuine" multicast algorithms [13].

[1] Local total order does not prevent processes in intersecting destination sets to deliver messages in different orders.

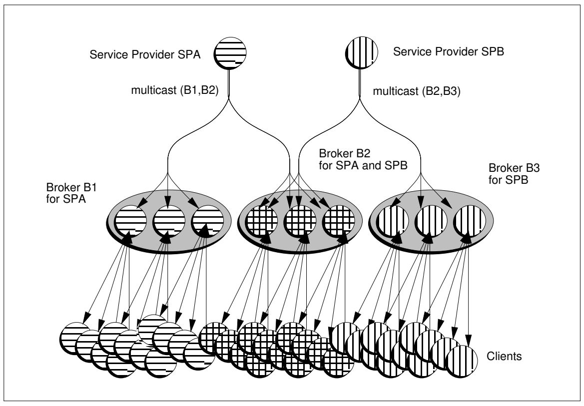

Figure 1. Open multicast model

Roughly speaking, the participant minimality property requires that *the execution of the algorithm implementing the total order multicast of a message* m *to a destination set* $\mathsf{Dst}(m)$ *involves only the sender process, and the processes in* $\mathsf{Dst}(m)$. The same authors proposed a total order algorithm that satisfies this property. However, their algorithm relies on causal order delivery of messages, which implicitly means that the size of the messages exchanged is a function of the total number of processes in the system (the message size can be reduced at the cost of increasing the synchrony of the system).

We go a step further in this paper by introducing the property of *message size minimality*, which requires that *the size of the messages in the protocol implementing the total order multicast of a message* m *to a destination set* $\mathsf{Dst}(m)$*, is a function of the number of processes in* $\mathsf{Dst}(m)$ (i.e., and not a function of the total number of processes in the system). We present an algorithm, named SCALATOM, which satisfies both participant minimality and message size minimality properties.

Our SCALATOM algorithm is also scalable in other dimensions. First, a process can multicast a message to one or more groups in the system without being forced to join those groups; thus group size does not grow with the number of clients. Second, SCALATOM relies on underlying components that are also scalable, namely a reliable multicast and a consensus protocol, and SCALATOM does not depend neither on specific implementations of these components, nor on implicit inter-dependencies among them (we also discuss a less modular, but optimized, variant of our algorithm). Finally, SCALATOM does not make strong assumptions on the underlying communication system, namely it does not assume neither reliable failure detection nor causal (or even fifo) ordering of messages.

None of the total order multicast algorithms we know about reaches SCALATOM's degree of full scalability, while preserving a reasonable latency. Other total order multicast algorithms either (*i*) implement just *local total order*, e.g., [4, 15, 24, 22], or (*ii*) do not satisfy the participant minimality property, e.g., [1, 6, 5, 7, 9, 19], or (*iii*) do not satisfy the message size minimality, e.g., [13], or (*iv*) require a group membership service that can exclude correct processes from the groups [3], or (*v*) require a failure detector that never makes any false suspicion about the crash of processes, which is known to be unrealistic in large scale systems that can be subject to link failures, e.g., [12, 17, 25]. To our knowledge, the algorithm that is closer to ours the recent one proposed by Fritzke et al. [18]. However, this algorithm introduces a higher latency than SCALATOM.

1.3. Overview of the paper

The rest of the paper is structured as follows. We first present the design and structure of SCALATOM in Section 2. Section 3 gives the system model and recalls some definitions. Section 4 describes the SCALATOM algorithm. Section 5 discusses related work, and compares the cost of SCALATOM with that of previous total order multicast algorithms. Section 6 concludes the paper. The correctness proofs of the SCALATOM algorithm are in the Appendix.

2. Achieving scalable atomic multicast

This section presents the design principles we have followed in the conception of the SCALATOM algorithm. We start by introducing the intuition behind our algorithm, making the bridge to previous algorithms that follow the same principle. Then we point out the modular structure of SCALATOM.

2.1. Principle of SCALATOM

SCALATOM is inspired by the Multicast Total Order Algorithm (MTO) of Guerraoui and Schiper [13][2], which, in turn, was inspired by an algorithm of Skeen [25]. Skeen's algorithm satisfies the participant minimality property, but is only correct under the assumption of a perfect failure detector. This assumption is unrealistic in most practical systems because a process that is slow (or with which the connection is temporary broken) can be timed-out and falsely suspected to have crashed by other processes. MTO does not assume a perfect failure detector (nor a membership service), but requires causally ordered messages. This assumption restricts the scalability of MTO, as the size of causal information that needs to be piggybacked with each protocol message can be of the order of n^2, where n is the total number of processes in the system (i.e., MTO does not satisfy

[2]This first version of MTO has an error that has been corrected in [14].

the message size minimality property)[3].

The motivation of this paper was to design an algorithm which does not depend on causal order but which still ensures the same useful properties as MTO. Although our SCALATOM algorithm shares some similarities with Skeen's algorithm and MTO, there are also significant differences. Before clarifying these differences, we first start by describing the basic structure, common to all protocols.

The basic idea of those protocols consists in defining a unique sequence number for every message that is multicast, and delivering the messages according to their sequence numbers. The three algorithms perform the following steps.

1. Messages are reliably disseminated to all addressees.

2. Participant processes (sender and addressees of messages) receive multicast messages, assign timestamps to these messages, and keep these messages in a *pending buffer*.

3. For each message m, participants agree on a unique *Sequence Number*, noted SN(m), which is computed using, as input for the agreement, the timestamps assigned to m. Messages which have been assigned a SN are moved from the pending queue to a *delivery queue*, where they are inserted in the order of their sequence number.

4. For each message m, participants compute the set of messages that can potentially be assigned a SN lower than SN(m). We name this set, the *Potential Predecessor Set* of m, noted PPS(m).

5. Finally, messages are delivered in the order of their sequence number SN. The message m at the head of the delivery queue is delivered, as soon as all messages in PPS(m) have already been assigned a sequence number and inserted in the delivery queue.

We now briefly describe how each of the three algorithms (i.e., Skeen's algorithm, MTO and SCALATOM) perform these steps.

Skeen's algorithm. Skeen's algorithm is simple, thanks to the very strong assumptions on which it is based (the original version assumes a failure-free system, and a fault-tolerant version requires a perfect failure detector). According to this algorithm, timestamps are incremented every time a message is sent or received. For every message m, all correct addressees must participate in the agreement on a sequence number SN (hence the need for a perfect failure detector) and SN is the maximum timestamp of *all* timestamps assigned to m by the correct addressees of m. The potential predecessor set (PPS) for a message m is trivially computed when the decision on the sequence number SN (m) is reached: PPS(m) is the set of messages in the pending queue with a timestamp lower than SN(m).

MTO. In the MTO algorithm of Guerraoui and Schiper, timestamps must also be updated by all messages, including the ones used to reach agreement. For every message m, only a majority of correct addressees needs to compute the sequence number SN(m), and the failure detector does not need to be perfect. A consensus-like protocol is used to ensure that all participants agree on the same sequence number. The potential predecessor set PPS(m) for a message m is also computed when the decision on SN(m) is reached: PPS(m) is the (complete) set of messages in the pending queue (and not only those in the pending queue with a lower timestamp than SN(m)). The MTO algorithm depends on causal order to ensure that the pending queue contains PPS(m) when the decision is reached.

SCALATOM. In our SCALATOM algorithm, timestamps are only incremented when messages that are multicast are first received or when a new sequence number is computed. The sequence numbers are computed exactly as in MTO. However, the computation of PPS is quite different. Since SCALATOM does not depend on causal order, there is no guarantee that, for every message m, the pending queue contains PPS(m) when SN(m) is decided. Thus, SCALATOM requires an additional step to accurately compute PPS(m). The computation of PPS(m) is performed locally inside a group, based only on gossip information exchanged among processes belonging to the same group.

2.2. The structure of SCALATOM

Fault-tolerance is achieved in SCALATOM using several sub-protocols (Fig. 2): (1) a reliable multicast used to issue the message (*R-multicast*), (2) a send primitive to disseminate timestamps and gossip information, (3) a uniform consensus protocol noted *Consensus*, used to compute the sequence number, and (4) a local function to compute the potential predecessor set of each message PPS(m) based on gossip information.

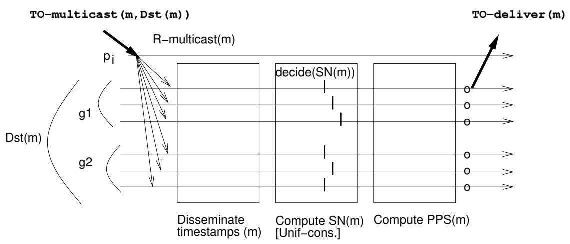

Figure 2. Principle of the SCALATOM algorithm

[3]The conservative implementation of causal order presented in [4] would not lead to violate the message size minimality property, but requires a group membership service. With unreliable failure detection, a group membership service might force correct processes to crash.

We believe the use of a modular approach in the implementation of a total order algorithm is also a key factor to achieve scalabity [16]. A modular construction based on the composition of encapsulated components allows the protocol to be optimized for different environments, simply by adapting the implementation of sub-protocols. SCALATOM uses a modular approach since the correctness of the protocol does not depend on a specific implementation of reliable multicast or uniform consensus. These sub-protocols do not need neither to share any common variable. Figure 3 illustrates the composition of modules required to implement SCALATOM.

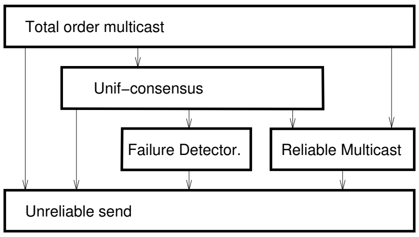

Figure 3. SCALATOM building blocks

3. System model and definitions

Before presenting the algorithm in details, we first describe the system model and define the companion services that are required to support SCALATOM.

Asynchronous system We consider a distributed system composed of a finite set of processes $\Omega = \{p_1, p_2, \ldots, p_n\}$ completely connected through a set of channels. Communication is by message passing, asynchronous and eventually reliable. Asynchrony means that there is no bound on communication delays, nor on process relative speeds. An eventual reliable channel ensures that a message sent by a process p_i to a process p_j is eventually received by p_j, if p_i and p_j are correct (i.e., do not fail)[4]. An eventual reliable channel can be implemented by retransmitting lost or corrupt messages. Processes fail by crashing (we do not consider Byzantine failures). A correct process is a process that does not crash in an infinite run.

Failure detectors Given the impossibility of reaching consensus in asynchronous systems [11], alternative system models have been defined: partially synchronous [8], timed asynchronous [10], and asynchronous augmented with failure detectors[5]. In this paper we follow the later model. We consider failure detectors in class $\diamond S$ (*Eventually Strong*), the *weakest* class of failure detectors that allow to solve consensus and atomic broadcast/multicast problems [5]: such failure detectors can make an infinite number of false suspicions. Failure detectors are required to solve consensus but are not otherwise used in the SCALATOM algorithm.

[4]This does not exclude link failures, if we require that any link failure is eventually repaired.

TO1 - Uniform Agreement: Consider *TO-multicast*$(m, \mathsf{Dst}(m))$. If a process in $\mathsf{Dst}(m)$ (correct or not) has *TO-delivered*(m), then every correct process in $\mathsf{Dst}(m)$ eventually *TO-delivers*(m).

TO2 - Termination: If a correct process *TO-multicasts*$(m, \mathsf{Dst}(m))$, then every correct process in $\mathsf{Dst}(m)$ eventually *TO-delivers*(m).

TO3 - Uniform Total order: Let m_1 and m_2 be two messages that are *TO-multicast*. We note $m_1 < m_2$ if and only if a process (correct or not) *TO-delivers* m_1 before m_2. Total order ensures that the relation $<$ is acyclic.

Table 1. Total order properties

Total order multicast to multiple groups We assume that the set Ω of processes is partitioned into non-intersecting groups g_i, and messages are multicast to groups. Given a message m, we note $\mathsf{Dst}(m)$ the set of process groups to which m is multicast. Total order multicast is defined on the set Ω by the primitives (1) *TO-multicast*$(m, \mathsf{Dst}(m))$ which issues m to $\mathsf{Dst}(m)$, and (2) *TO-deliver(m)* which is the corresponding delivery of m. When a process p_i executes *TO-multicast*$(m, \mathsf{Dst}(m))$ (resp *TO-deliver(m)*), we say that p_i *"TO-multicasts m"* (resp *"TO-delivers m"*). The properties of the primitive *TO-multicast* [15] are listed in Table 1.

Reliable multicast. We assume the existence of a *reliable* multicast primitive, denoted *R-multicast(m,Dst(m))*, and the corresponding reliable delivery primitive, denoted *R-deliver(m)*. The primitive *R-multicast(m,Dst(m))* satisfies the following two properties: (1) (agreement) if a process in Dst(m) (correct or not) has *R-delivered*(m), then every correct process in Dst(m) eventually *R-delivers*(m); (2) (validity) if a correct process *R-multicasts(m,Dst(m))*, then every correct process in Dst(m) eventually *R-delivers*(m) [15]. An example of a simple implementation of reliable multicast is given in [5].

Qualified majority. Let f_i be the maximum number of processes of group g_i that can crash, and assume $f_i < |g_i|/2$ for every group g_i (i.e., we assume a majority of correct processes in each group). Given *TO-multicast*$(m, Dst(m))$, we define a *qualified majority* of Dst(m), noted $QM(Dst(m))$, as any subset of Dst(m) that contains a majority of processes of every group g_i in Dst(m). For example, if Dst(m) $= \{g_1, g_2\}$, $g_1 = \{p_1, p_2, p_3\}$ and $g_2 = \{p_4, p_5, p_6\}$, then $\{p_1, p_2, p_4, p_5\}$ is a qualified majority of Dst(m), whereas $\{p_1, p_2, p_3, p_4\}$ is not.

Consensus. We finally assume the existence of a *consensus* function, which solves the uniform consensus problem [5]. The problem is defined over a set of processes Π, each proposes an initial value, and has to decide on a final value, such that (1) agreement: no two processes decide differently, (2) termination: every correct process eventually decides, and (3) non-triviality: the value decided is one of the values proposed.

4. The SCALATOM algorithm

This section describes SCALATOM. We start by describing the variables and data structures used in the algorithm, and then we describe each step of the algorithm. The lemmas and propositions that are used to state and prove the correctness of the algorithm can be found in the extended report [23].

4.1. Variables and data structures

In order to execute the algorithm, each process p_j maintains the following variables and data structures.

A logical clock is used to timestamp R-delivered messages. The clock is incremented every time a message is R-delivered (deliver event) and every time a sequence number is decided (decide event).

Three buffers are used to store TO-multicast messages until they are TO-delivered. When a message m is R-delivered, it is first inserted in a *pending* buffer. Message m is kept in the pending buffer until its sequence number SN(m) is computed. Then m is moved to a *compute-PPS* buffer. Finally, when $PPS_j(m)$ is computed, the message m is moved from the *compute-PPS* buffer to a *delivery* queue, where m is inserted in the order of its sequence number.

The sequence of pairs (event, associated timestamp) of a process p_i, is named the process *history*, denoted $history(p_i)$. Only two types of events need to be logged in the process history: R-deliver and decide events. The SCALATOM algorithm requires each process to store a subset of its own history as well as a subset of the delivery histories of other correct processes *from the same group*. To achieve this, processes gossip histories among each others (note that this information is local to each group).

```
// ts_j (timestamp, incremented on R-deliver and decide)
1. when [TO-multicast(m, Dst(m))]
   R-multicast (m, Dst(m)) ;
2. when [R-deliver_j (m)]
   rec-ts_j (m) ← ts_j ;
   ts_j ← ts_j + 1;
   send rec-ts_j (m) to Dst(m) ;
   add m to pending buffer ;
   add (R-deliver_j (m), ts_j) to history(p_j) ;
3. when [received rec-ts_k (m) from a qualified majority of Dst(m)]
   prop_j (m) ← max of all rec-ts_k (m) received ;
   start consensus (prop_j (m), Dst(m)) ;
4. when [Consensus (prop_j (m), Dst(m)) terminates]
   SN(m) ← consensus (prop_j (m), Dst(m)) ;
   ts_j ← max (ts_j, SN(m)) +1 ;
   add (decide(SN(m)), ts_j) to history(p_j) ;
   move m from pending buffer to compute-PPS buffer ;
   send history(p_j) to local group ;
5. when [history(p_k, SN(m)) received from a majority of local group ]
   PPS(m) ← compute-PPS(maj. history(p_k, SN(m))) ;
   move m from compute-PPS buffer to delivery queue ;
   deliver-in-order (SN(m), PPS(m)) ;
```

Figure 4. SCALATOM execution steps

4.2. Protocol steps

The algorithm is illustrated in Figure 4. We describe each of the protocol steps below.

1. Dissemination step. This step is initiated when *TO-multicast*$(m, Dst(m))$ is executed by a process p_i: the message m is *R-multicast* to Dst(m). Hence, if the sender is correct, or if one addressee R-delivers the message, all correct addressees of m also R-deliver the message. This also implies that the correct members of every group R-deliver the same set of messages, although not necessarily in the same order.

2. Timestamp dissemination step. When the multicast message is R-delivered at some addressee p, p performs the following actions: p assigns a timestamp to the R-delivered message, updating the logical clock accordingly; p sends the timestamp to all processes in Dst(m); then p stores the R-delivered message in the pending buffer.

3. Sequence number computation step. The process p then waits until it has collected the timestamps from a qualified majority of Dst(m) in order to invoke the *Consensus* function to compute the sequence number $SN(m)$. When the consensus terminates, the sequence number SN(m) has been computed for m. The local timestamp is updated accordingly, and the message m is removed from the pending buffer and inserted in the compute-PPS buffer.

4. Potential predecessor set computation step. The PPS(m) is computed locally based on gossip information. Thus, an additional communication step is performed among group members to exchange process histories. Note that it is trivial to implement gossip in such a way that each gossip message only carries new entries and not complete histories[5].

Let $history(p_i, n)$ denote the subset of process p_i's history containing all R-delivery events with a timestamp lower than n. The potential predecessor set of message m is computed locally, as soon as the process has collected $history(p_j, SN(m))$ from a majority of group members. The PPS(m) is simply the set of all messages in $history(p_j, SN(m))$ from this majority. This ensures that PPS(m) contains all messages of which sequence numbers can be lower than $SN(m)$. It should be emphasized that messages that have been already TO-delivered do not need to be inserted in PPS(m); any efficient implementation of the protocol should exclude these messages from PPS computation to avoid unnecessary tests. However, for simplicity of presentation, we omit this optimization. When PPS(m) is computed, m is removed from compute-PPS buffer and inserted in the delivery queue, ordered according to $SN(m)$.

5. Delivery step. Finally, a message is TO-delivered when the following condition is met: the message m is at the head of the delivery queue *and* all messages in PPS(m) have been assigned a sequence number *and* either (i) these sequence numbers are higher than SN(m) *or* (ii) these sequence numbers are lower than SN(m) and the associated messages have been TO-delivered locally.

4.3. Scalability of SCALATOM

Consider a message m that is TO-multicast to Dst(m). The Implementation of R-multicast$(m, Dst(m))$ only involves the sender and the processes in Dst(m) [5]. The only message that is R-multicast is m. Consensus is defined only for processes in Dst(m) [5], thus it only involves processes in Dst(m), and the size of the messages exchanged depends only on Dst(m). Finally, the last communication step of the protocol does involve only processes of the same group. Hence SCALATOM satisfies the participant minimality and message size minimality properties.

5. Related work

In this section, we give an overview of other total order algorithms and we compare their scalability and performance characteristics with those of SCALATOM. Given the myriad of total order algorithms that can be found in literature, this comparison must be necessarily selective.

We have chosen three approaches that can be considered archetypes of many popular total order protocols: the sequencer-site approach, the rotating token-approach, and the symmetric approach. Although these three classes of protocols are based on very different system assumptions (e.g., broadcast instead of multicast, perfect failure detectors or membership protocol), they give an approximate idea of what type of costs can be expected. We also compare SCALATOM with the Chandra-Toueg's algorithm [5], with the MTO algorithm from Guerraoui and Schiper [13], and with the algorithm of Fritzke et al [18]. These three protocols enable a more accurate comparison as they are all multicast algorithms based on unreliable failure detectors of class $\diamond\mathcal{S}$ [5], and none is based on a membership service.

We analyze the cost of these algorithms in terms of number and size of messages exchanged, and number of communication steps needed to TO-deliver a message[6]. We consider in our analysis only the best case scenario, i.e. runs with no failure and no failure suspicion. This is the most frequent case in practice.

Sequencer-site algorithms. This class of total order algorithms is based on the idea of a specific process (sequencer) assigning a sequence number to all messages [19]. A single communication step is required in the optimal case where the sequencer is also the source of the message, and two steps in all other cases. This algorithm is inherently non-scalable, even for medium size system, due to the overload of the sequencer. For fault-tolerance, the failure of the sequencer must be reliably detected. Additionaly, system contamination may occur. Contamination can be prevented if sequence numbers are disseminated using reliable uniform multicasts but this obviously adds additional communication steps to the algorithm.

Rotating-token algorithms. Algorithms in this class are similar to sequencer-site algorithms, but rotate the role of sequencers among several processes [6, 4, 1]. This provides load balancing and avoids network contention when shared links are used (for instance in LANs) [21]. Unfortunately, in the worst case, a message may wait for complete rotation of the token before it gets assigned a sequence number: the number of communication steps is thus a function of the size of the system, which is not scalable.

Symmetric algorithms. These algorithms are based on Lamport's total order algorithm [20], i.e., messages are delivered according to the order defined by the timestamps assigned at multicast time [7, 1, 9]. In most cases, in order to deliver a message, a process p simply waits until another message with equal or higher timestamp is received

[5]Groups can also piggyback gossip messages with other gossip or data messages.

[6]This measure is less ambiguous than the usual number of "phases". To give an example, the classical two phase commit protocol (2PC) has 3 communication steps [2]: (1) *vote request* sent from the coordinator to the participants, (2) *reply* of the participants sent to the coordinator, and (3) *decision* sent by the coordinator to the participants.

from every process that can interact with p (FIFO channels are often assumed). An interesting variant described in [7] requires a message to be received from just a majority of the interacting processes but requires the construction of a (causal) message graph. Total order can be established in a single communication step when all processes broadcast simultaneously [22], and in two steps in all other cases. The number of messages required is in the order of the group size if a *closed* model is assumed, i.e., if only members of a group can multicast messages to the group [9]. In an open model, a symmetric algorithm might force each process to interact with all other processes in the system and thus, the number of messages is a function of the system size.

Chandra and Toueg's algorithm. This algorithm consists of (1) reliably broadcast of message m, followed by (2) the execution of a consensus. Chandra-Toueg's consensus algorithm based on $\diamond\mathcal{S}$ requires 3 communication steps. Thus, in the best case, 4 communication steps are required to run the total order broadcast algorithm. In terms of the number of messages, Chandra and Toueg's algorithm requires $(n-1)^2$ for the reliable broadcast and $2(n-1)+(n-1)^2$ to run consensus, for a total of $2(n-1)^2+2(n-1)$.

Guerraoui and Schiper's algorithm. The MTO algorithm consists of (1) reliably multicast of message m to $\mathsf{Dst}(m)$, then (2) the exchange of timestamps among $\mathsf{Dst}(m)$, and finally (3) the consensus. Thus, with regard to Chandra and Toueg's algorithm, MTO has an additional communication step (for a total of 5) but generates less messages due to the participant minimality property. Let d denote the number of addressees of a message; MTO generates $d+(d-1)^2$ messages in the initial R-multicast, $d(d-1)$ messages to exchange timestamps plus $2(d-1)+(d-1)^2$ in the consensus, for a total of $3(d-1)^2+4(d-1)$ messages. The size of messages is a function of the total number of processes in the system due to the causal order requirement and is in the order of n^2.

SCALATOM algorithm. Our algorithm follows the same structure as the MTO algorithm, the main difference being the technique used to compute the potential predecessor set of a given message. Even though the computation of PPS is performed locally, it is based on the construction of a view of the other processes' histories. These views are constructed by the exchange of gossip messages that can trivially be piggybacked with the messages sent to exchange timestamps, so this mechanism is not a source of additional messages. Unfortunately, like the symmetric algorithms mentioned above, without the additional assumptions that messages are periodically sent to every group, liveness cannot be ensured unless an additional communication step is performed. Thus, SCALATOM requires 6 communication steps and $3(d-1)^2+4(d-1)+(d-1)^2$ messages. However, SCALATOM satisfies the message size minimality property, which means that message size is a function of the number of participating processes.

Fritzke et al. algorithm. This algorithm offers an elegant solution to the total order multicast problem that also satisfies our minimality constraints [18]. The proposed solution does not relies on the computation of the PPS. Instead, an additional consensus is run to synchronize the timestamps. Thus, the consensus protocol needs to be executed twice for each message, resulting in a high latency solution: at least 7 communication steps are needed to terminate the protocol.

Tree-based algorithms The algorithm presented by Garcia-Molina and Spauster [12], and its extension described in [17], satisfy the participant minimality property but both require reliable failure detectors. Additionally, the number of steps required to perform a multicast depends on the multicast propagation tree. Thus, it is hard to make a simple comparison with our approach.

Trade-off. Table 2 summarizes the numbers given above, for the case of a multicast sent to d processes in a system of n processes ($d \leq n$). All n processes take part to the protocol in the case of a broadcast, whereas only d processes take part to the protocol in the case of a multicast. The figure illustrates the interesting trade offs (i) between *broadcast* with 4 communication steps (which does not require causal order delivery), where all n processes take part to the protocol, (ii) *multicast* with 5 communication steps, where only the d processes in $\mathsf{Dst}(m)$ take part to the protocol ($d \leq n$), but with the cost of causal delivery (larger messages), (iii) *multicast* also with 6 communication steps, where only the d processes in $\mathsf{Dst}(m)$ take part to the protocol ($d \leq n$) and not requiring causal delivery. Other solutions [18] exhibit an even higher latency.

6. Concluding remarks

The paper has discussed the issue of designing a scalable total order multicast algorithm, and defined desirable properties for a scalable algorithm, such as participant minimality and message size minimality. We presented a new fault-tolerant total order multicast algorithm, named SCALATOM, that satisfies these properties, and that does not depend on reliable failure detection. To our knowledge, this is the first total order multicast to reach such a high degree of scalability with low latency overhead.

SCALATOM has a modular structure that promotes a design methodology based on the composition of encapsulated micro-protocols. The latency of SCALATOM can be reduced (we can drop one communication step) if we assume that new messages are always multicast, or if we

	Com. Steps	Nb. of Mesg.	Mesg. Size
Sequencer-site (broadcast,PFD)	2 [1]	$2(n-1)$	const.
Rotating-token (broadcast,PFD)	1 to n-1	$(n-1)$ $[+n]$	const.
Symmetric (broadcast,PFD)	2 [1]	$(n-1)$ $[+n(n-1)]$	const.
Chandra-Toueg (broadcast,$\diamond\mathcal{S}$)	4	$2(n-1)^2+2(n-1)$	$f(n)$
MTO (multicast, causal,$\diamond\mathcal{S}$)	5	$3(d-1)^2+4(d-1)$	$f(n^2)$
SCALATOM (multicast,$\diamond\mathcal{S}$)	6	$3(d-1)^2+4(d-1)$ $[+(d-1)^2]$	$f(d)$
Fritzke-et al. (multicast,$\diamond\mathcal{S}$)	≥ 7	$4(d-1)^2+4(d-1)$	$f(d)$

Table 2. Comparative analysis

enable variable sharing between the sub-protocols used in SCALATOM. We are currently working on alternative optimizations that would preserve the scalable (and hence modular) nature of SCALATOM, and more generally on better understanding the trade-offs between scalability and latency.

References

[1] Y. Amir, L. Moser, P. Melliar-Smith, D. Agarwal, and P.Ciarfella. The Totem Single-Ring Ordering and Membership Protocol. *ACM Trans. on Computer Systems*, 13(4):311–342, November 1995.

[2] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Distributed Database Systems*. Addison-Welsey, 1987.

[3] K. Birman and T. Joseph. Reliable Communication in the Presence of Failures. *ACM Trans. on Computer Systems*, 5(1):47–76, February 1987.

[4] K. Birman, A. Schiper, and P. Stephenson. Lightweight Causal and Atomic Group Multicast. *ACM Trans. on Computer Systems*, 9(3):272–314, August 1991.

[5] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of ACM*, 34(1):225–267, 1996. A preliminary version appeared in the *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 325–340. ACM Press, August 1991.

[6] J. M. Chang and N. Maxemchuck. Reliable Broadcast Protocols. *ACM Trans. on Computer Systems*, 2(3):251–273, August 1984.

[7] D. Dolev, S. Kramer, and D. Malki. Early Delivery Totally Ordered Broadcast in Asynchronous Environments. In *IEEE 23rd Int Symp on Fault-Tolerant Computing (FTCS-23)*, pages 544–553, June 1993.

[8] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–323, April 1988.

[9] P. Ezhilchelvan, R. Macedo, and S. Shrivastava. Newtop: A Fault-Tolerant Group Communication Protocol. In *IEEE 15th Intl. Conf. Distributed Computing Systems*, pages 296–306, May 1995.

[10] C. Fetzer and F. Christian. On the possibility of consensus in asynchronous systems. *Pacific Rim Int. Conference on Fault-Tolerant Systems*, December 1995.

[11] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32:374–382, April 1985.

[12] H. Garcia-Molina and A. Spauster. Ordered and Reliable Multicast Communication. *ACM Trans. on Computer Systems*, 9(3):242–271, August 1991.

[13] R. Guerraoui and A. Schiper. Total order multicast to multiple groups. In *IEEE 17th Intl. Conf. Distributed Computing Systems*, pages 578–585, May 1997.

[14] R. Guerraoui and A. Schiper. Genuine atomic multicast in asynchronous systems. Technical Report 98/273, EPFL, March 1998.

[15] V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.

[16] M. Hiltunen and R. Schlichting. An approach to constructing modular fault-tolerant protocols. In *Proceedings of the 12th Symposium on Reliable Distributed Systems*, pages 105–114, Princeton, New Jersey, Oct. 1993. IEEE.

[17] X. Jia. A Total Ordering Multicast Protocol Using Propagation Trees. *IEEE Trans. Parallel & Distributed Syst.*, 6(6):617–627, June 95.

[18] U. F. Jr., P. Ingles, A. Moustefaoui, and M. Raynal. Fault-tolerant total order multicast to asynchronous groups. Technical Report 1162, IRISA, 1998.

[19] M. F. Kaashoek, A. S. Tanenbaum, S. F. Hummel, and H. E. Bal. An Efficient Reliable Broadcast Protocol. *Operating Systems Review*, 23(4):5–19, October 1989.

[20] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, July 78.

[21] L. Moser, Y. Amir, P. Melliar-Smith, and D. Agarwal. Extended Virtual Synchrony. In *IEEE 14th Intl. Conf. Distributed Computing Systems*, pages 56–67, June 1994.

[22] L. Rodrigues, H. Fonseca, and P. Veríssimo. Totally ordered multicast in large-scale systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, pages 503–510, Hong Kong, May 1996. IEEE.

[23] L. Rodrigues, R. Guerraoui, and R. Schiper. Scalable atomic multicast. Technical Report RT98-2, FCUL/EPFL, 1998.

[24] L. Rodrigues and P. Veríssimo. xAMp: a Multi-primitive Group Communications Service. In *Proceedings of the 11th Symposium on Reliable Distributed Systems*, pages 112–121, Houston, Texas, Oct. 1992. IEEE.

[25] D. Skeen. Unpublished communication. Feb 1985. Referenced in *K. Birman and T.Joseph*. Reliable Communication in the Presence of Failures. ACM Trans. on Computer Systems, 5(1):47-76, February 1987.