Total Order Broadcast on Pervasive Systems

Luiz Angelo Steffenel, Manuele Kirsch Pinheiro, Yolande Bebers

▶ **To cite this version:**

Luiz Angelo Steffenel, Manuele Kirsch Pinheiro, Yolande Bebers. Total Order Broadcast on Pervasive Systems. ACM Symposium in Applied Computing, Mar 2008, Fortaleza, Brazil. pp.to be announced. hal-00261438

HAL Id: hal-00261438
https://hal.science/hal-00261438

Submitted on 7 Mar 2008

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Total Order Broadcast on Pervasive Systems

Luiz Angelo Steffenel
CReSTIC - SYSCOM Team
Department of Math. and Computer Sciences
Université de Reims Champagne-Ardenne
BP 1039, F-51687 Reims Cedex 2, France
Luiz-Angelo.Steffenel@univ-reims.fr

Manuele Kirsch Pinheiro,Yolande Bebers
Department of Computer Science
Katoliek Universiteit Leuven
Celestijnenlaan 200a
B-3001 Leuven, Belgium
Manuele.KirschPinheiro@cs.kuleuven.be
Yolande.Bebers@cs.kuleuven.be

ABSTRACT

Total Order Broadcast protocols are important tools to ensure coherence across distributed systems. Contrarily to classical distributed systems, pervasive systems bring important constraints related to the performance and reliability of the network and the availability of the devices (laptops, PDAs and cellular telephones). We propose in this paper a self-stabilizing group membership service that helps a token-based Total Order Broadcast protocol to progress in a volatile environment. This group membership service is organized in two hierarchical levels so that unstable nodes are kept in the group without interfering with the Total Order Broadcast protocol. As a result, we avoid expensive membership view changes while keeping the coherence among the nodes.

Categories and Subject Descriptors

D.4.5 [**Operating Systems**]: Reliability; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed applications*; H.4.3 [**Information Systems Applications**]: Communications Applications

General Terms

Algorithms, Reliability

Keywords

Total Order Broadcast, Group Membership, Self-stabilization

1. INTRODUCTION

Total Order Broadcast, also known as Atomic Broadcast [12] is one of the essential building blocks of fault tolerant distributed systems. Thanks to Total Order Broadcast algorithms, a global coherent view of a data set (a document, for example) can be provided, as messages that modify this data set are received by all correct processes and executed in the same order.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil
Copyright 2008 ACM 978-1-59593-753-7/08/0003 ...$5.00.

When working on pervasive systems, however, the tolerance to frequent disconnections becomes one of the key performance aspects. Indeed, pervasive systems are confronted to constraints such as limited battery lifetime of mobile devices (laptops, PDAs, cellular phones) and frequent disconnections of wireless networks (WiFi, GPRS...). Consequently, these systems suffer from nodes volatility and network coverage problems.

In this paper, we address the problem of delivering messages in a global total order for a distributed application running in a pervasive environment. We propose a self-stabilizing membership strategy that helps a *token*-based Total Order Broadcast algorithm to circumvent blocking situations due to the frequent disconnections of the nodes.

This paper is organized as follows: In Section 2 we describe the distributed environment we are dealing with. Section 3 introduces the Total Order Broadcast problem and its implementation issues. In Section 4 we study both membership and failure detection requirements for operating in a dynamic pervasive system, presenting our membership strategy as well as some experimental results obtained on a cluster environment. Finally, Section 5 reviews works related to the Total Order Broadcast problem and Section 6 presents the conclusions of this work.

2. ENVIRONMENT DESCRIPTION

We consider the problem of a distributed agreement (the Total Order Broadcast) in the context of a pervasive system composed of mobile devices interconnected by a mix of standard infrastructures (fixed networks) and wireless networks (Fig. 1). These mobile nodes are equipped with standard and/or wireless communication interfaces that allow them to move at will, as well as allowing them to connect over a fixed structure (as in the case of laptop computers). In such an environment, nodes that are located at the boundaries of the wireless coverage zone may be out of reach from time to time. Also, mobile devices that have low power capacities may disconnect themselves to save battery power.

The combination of node mobility and a wireless environment can result in topologies subject to rapid and unpredictable changes. As a result, traditional agreement techniques cannot fulfill the requirements of this environment, thus requiring new solutions that integrate fault-tolerance and self-stabilization in order to prevent the protocol from blocking indefinitely. We assume that while nodes in a pervasive environment may disconnect regularly, eventually the network will stabilize so that a number of nodes remain con-

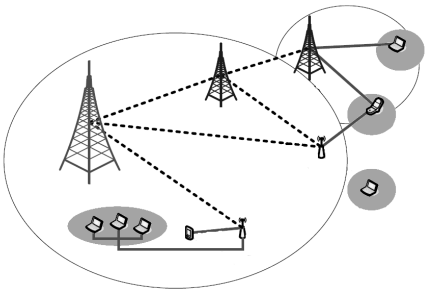

Figure 1: Topology with mobile devices

nected long enough to allow a Total Order Broadcast protocol to progress.

Applications built on the top of such pervasive environments often need to ensure reliable communication among the nodes involved in order to prevent the loss of important data. For example, let us consider a synchronous collaborative editor [16], a CAD project, for example. In this kind of application, several users work on the same document and actions performed in one node have to be transmitted to all nodes and performed in the same order. For instance, if an insert action, performed in a given node, is followed by a remove action, performed in a second node, this order must be respected in all collaborating nodes and no messages should be lost so that the same resulting document will be obtained everywhere.

We propose in this paper the composition of an efficient Total Order Broadcast algorithm and a Group Membership management system specifically tailored for a volatile network. We consider a set of n processes $\Pi = p_0, ..., p_{n-1}$ in a pervasive environment. The processes interact through message exchange over unreliable channels where processes may fail by crashing (i.e., we do not consider Byzantine faults). In an asynchronous distributed system, however, it is hard to distinguish between a wrongly suspected process and a crashed one (also known as the FLP Impossibility [9]). To circumvent this limitation, the system is augmented with unreliable failure detectors [2], which can be used to reach an agreement in spite of wrong suspicions.

3. PROBLEM DEFINITION

In this paper we focus on the Total Order Broadcast problem. A Total Order Broadcast ensures that processes in a distributed system deliver messages in the same order, which is essential for implementing services that require coherence between processes such as distributed databases or collaborative edition. This problem can be defined by four properties [6], as presented below:

VALIDITY - If a ***correct*** process (a process is called *correct* only if it does not crash during the entire execution, although even a correct process can be incorrectly suspected of crashing) broadcasts a message m to a list of processes Π, then some ***correct*** process in Π eventually delivers m to the application.

AGREEMENT - If a ***correct*** process delivers a message m, then all ***correct*** processes in Π eventually deliver m.

INTEGRITY - For any message m, every ***correct*** process p delivers m at most once and only if (1) m was previously broadcast by *sender(m)* and (2) p is a process in the set Π.

TOTAL ORDER - If ***correct*** processes p and q both deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'.

To ensure these properties, Total Order Broadcast usually relies on a globally agreed sequence of messages (for example, by assigning sequence numbers to the messages) and delivering them to the application according to that order. Hence, a simple solution to provide Total Order would be to centralize the distribution of sequence numbers in a fixed process (the sequencer). This approach, however, is prone to failures, as the sequencer may fail or be disconnected. Indeed, most implementations in the literature rely on distributed agreement operations such as Consensus [2, 13], which make the processes agree on every message before delivering it to the application. Unfortunately, the Total Order Broadcast based on Consensus is an expensive operation in the context of pervasive systems. Indeed, a pervasive system subjected to frequent disconnections may require several Consensus rounds before a majority of processes agreed on a message order. In such a scenario, message delivery will be blocked until the gathering of a stable quorum.

An alternative to both techniques is the use of a *moving sequencer* strategy [3, 6, 8], which presents the performance of a sequencer-based implementation while preventing a single point of failure by rotating the role of sequencer among the nodes. Indeed, as the *moving sequencer* strategy does not rely on consensus to order messages, it can perform faster in a pervasive environment than consensus-based techniques. Furthermore, this strategy can be easily implemented using a token-passing algorithm (see Fig. 2): the process that holds the token ($p1$) is the only one that can assign sequence numbers to the messages. Sequence numbers are sent together with the token, in the same message. To become the next sequencer, a process must acquire all k previous messages, thus reinforcing the coherence between the processes (sometimes called k-resiliency, where k denotes the minimum number of coherent processes).

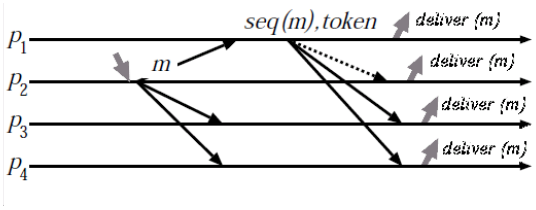

Figure 2: The *token-passing* mechanism

4. FAILURE HANDLING

Although fault-tolerant algorithms are designed to tolerate a certain number of process failures, sooner or later the accumulated number of failed or brutally disconnected processes (i.e., disconnected without notifying the group) may prevent the progress of the algorithm. In the case of consensus-based algorithms, this threshold depends on the majority quorum. In the case of token-based algorithms such as the *moving sequencer* algorithm, this threshold depends on the token passing mechanism. Indeed, as the token is passed in a logical ring, we must reconstruct the ring

whenever a failed node blocks the token passing or when the token is lost (the token holder has failed). In order to reconstruct the ring, also known as token list, we can use some techniques to manage the processes membership, such as the View Synchronous Communication model [5, 11].

The View Synchronous Communication (VSC, for short) protocol manages the creation and the maintenance of a set of processes during the execution. It is a distributed algorithm without a central coordinator, triggered when a failure detector [2] suspects a process. The successive memberships of a group are called *views* and the event by which a new view is provided to a process is called the *install* event. In VSC, broadcasts to members of the current view are delivered with some guarantees, the most notable being the *Same View Delivery* property [5]: if a process p delivers a message m in a view v, then every correct process ought to deliver the message m in the same view v.

Basically, a VSC membership view change gathers all correct processes in a new view v_{i+1}. To ensure that all processes in the new view are coherent, these processes deliver all queued messages before installing the view v_{i+1}. For the same reason, processes excluded from the view are forced to suicide (or to reconnect as a new process) to avoid incoherent copies of the data.

However, traditional membership algorithms are not designed to support group management in pervasive environments [1], as they usually assume that devices are connected by reliable networks and that disconnections are rare. To minimize the impact of *program-controlled crash* on a pervasive environment that is prone to frequent disconnections, we advocate the use a membership in two levels: suspected nodes are initially put into "quarantine", removing them from the token list but not from the group. Therefore, message sequencing is not blocked by suspected nodes and we still allow suspected nodes to recover and to keep updated.

The "two views" technique was initially proposed for the Primary-Backup replication algorithm [4]. This technique considers that each level of membership defines different types of views. *Ordinary views* (or simply *views*) are identical to the views of View Synchronous Communication, while *intermediate views* (or *i-views*) are installed between ordinary views.

If ordinary views are denoted by $v_0, v_1, ..., v_i$, the i-views between v_i and v_{i+1} are denoted as $v_i^0, v_i^1, \ldots, v_i^j, \ldots, v_i^{last}$. The intermediate view v_i^0 is equal to v_i and the last intermediate view v_i^{last} is equal to v_{i+1}. One important point is that the membership of all intermediate views $v_i^0, v_i^1, \ldots, v_i^{last-1}$ is the same as the membership of v_i, that is, they only differ in the order that processes are listed in the view. For example, $v_i = v_i^0 = \{p, q, r\}$, $v_i^1 = \{q, r, p\}$, etc. This allows a Primary-Backup replication algorithm to delay Membership Views as long as the first process in an intermediate view is able to act as Primary server. During the existence of the i-views $v_i^0, \ldots, v_i^{last-1}$ the ordinary view remains v_i.

This model allows us to minimize the *program-controlled crash* problem. Indeed, *ordinary views* are generated by suspicions resulting from failure detectors with conservative timeouts, while *i-views* are generated by suspicions resulting from aggressive timeouts. As all i-views from $v_i^0, \ldots, v_i^{last-1}$ are composed by the same set of processes, they do not force the crash of processes. This way, i-views avoid blocking situations while ordinary views ensure time-bounded buffering.

In the case of our problem, a simple shift in the membership order is not enough to ensure the liveness as the token is passed regularly, eventually blocking the algorithm. Instead, we move suspected processes into an "external set of processes", redefining i-views as a composed group **{"core members","external"}**. This way, messages are sent to the whole group while the token is passed only among **core members**. As a suspected process does not participate in the message sequencing (it is not in the core group), we minimize the probability of blocking the token passing (Fig. 3). As it still belongs to the view, it receives all broadcasts and can send messages to the group.

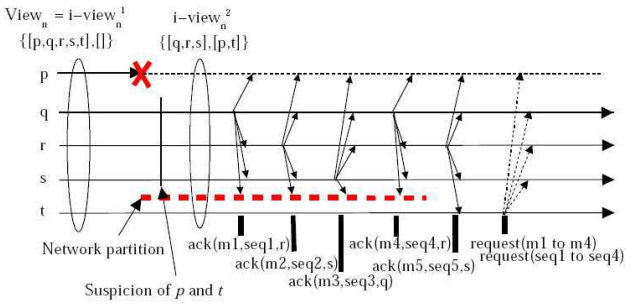

Figure 3: I-views and external members

Because external processes do not receive the token we cannot use the token passing to ensure message stability. Actually, as we don't know if an external process is correct or not, no assumptions can be made about it. For this reason, while Total Order properties (see section 3) must be ensured for the "core" processes, external processes are kept updated by requesting lost messages directly to the token holder. Consequently, the processes can cope with short disconnections, commonly found in wireless networks.

The efficiency of this technique relies on the network self-stabilization. Indeed, while a substantial part of the nodes may connect/disconnect regularly, a small group of "stable" nodes will eventually compose the "core group" of our view. As the token is passed only among stable nodes we drastically reduce the events that trigger a new Membership View.

However, messages cannot be stored forever. As stated by the *time-bounded buffering problem* [4], every message is eventually discarded from all output buffers. The time-bounded buffering is related to message stability: if process p knows that all processes in Π have either received m or crashed, then the message m becomes stable and p can remove it safely. However, Charron-Bost *et al.* [4] claim that the time-bounded buffering problem cannot be solved with realistic failure detectors. Instead, *program-controlled crash* is used to discard any inconsistent process q. Indeed, as q eventually crashes, there is no obligation for q to deliver m and thus p can safely discard m.

In the case of the Total Order Broadcast, a message can be discarded from the buffers as soon as it is delivered by a sufficiently large number of processes (the k-resiliency). As processes in the *external* group that reconnect after a long absence may be unable to acquire missing messages, they must commit "suicide" and reconnect with a different ID. When a new process *join* the group it triggers a Membership View change, becoming from that moment coherent with the other processes in the group.

Even reducing the probability of Membership View changes, several *i-view* changes may occur before reaching stability. In the next section (section 4.1) we present a light-weight algorithm for the *i-view* changes, reducing their cost. Later, in section 4.2, we evaluate the impact of different k-resiliency levels on the performance of the protocol.

4.1 Optimizing i-view changes

From the previous sections, we can define an algorithm for the Membership View Change, as presented in Algorithm 1.

Algorithm 1 Membership View Change algorithm

Upon suspicion of some process in Π_i
RBroadcast (reformation, i)
Upon R-Deliver (reformation, i) by p_k for the first time
1. send $seqQ_k$ to all /* sends the "unstable" list of messages */
2. $\forall p_i$, wait until receive $seqQ_i$ from p_i or p_i suspected
3. let $initial_k$ be the tuple (Π_k, $Msgs_k$) s.t.
- Π_k is the new token list with all processes that sent their $seqQ$
- $Msgs_k$ is the union of the $seqQ$ sets received
4. execute consensus among Π_i processes, with $initial_k$ as the initial value
5. let (Π, Msg) be the consensus decision
6. $stableQ \leftarrow Msg$, $seqQ \leftarrow \{\}$ /* as all processes get Msg, these messages are stable */
7. if $p_k \in \Pi$, then "install" Π as the next view Π_{i+1}
else suicide

Here, sequenced messages are kept in the sequenced queue $seqQ$ until they become stable. Once a process receives the sequenced queue from all processes that are not suspected, it can compute $Msgs_k$, the union of all received $seqQ$. It also can suggest a new token list based in the set of processes that answered the *reformation* message.

As processes agree both on the new token list and on the set of all unstable messages, all processes that acknowledge this decision have the same set of messages. This "uniform" knowledge can be considered on indicator of message stability and therefore these messages are ready to be delivered. Finally, as all processes share the same sequenced queue, any process can be the new sequencer and we can select, for example, the first process in the token list. Please note that suspected nodes excluded from the view are forced to commit suicide (Algorithm 1, line 7).

Although this algorithm is adapted to Membership View Changes, it is quite expensive when considering only *i-view* changes. By definition, i-views aim to avoid blocking the protocol, not to solve the time-bounded buffering problem. Indeed, message stabilization does not need to be implemented during an i-view change and we can optimize the i-view algorithm by skipping the exchange of sequenced queues. As a result, we have a new algorithm for i-view changes, as presented in Algorithm 2.

Algorithm 2 Optimized i-view changes

Upon suspicion of some process in Π_i
RBroadcast (i-view, i)
Upon R-Deliver (i-view, i) by p_k for the first time
1. send ack to all
2. $\forall p_i$, wait until receive ack from p_i or p_i suspected
3. let $initial_k$ contains Π_k s.t.
- Π_k is the new token list with all processes that sent ack
4. execute consensus among Π_i processes, with $initial_k$ as the initial value
5. let Π be the consensus decision
6. if $p_k \in \Pi$, then "install" Π as the next view Π_{i+1}

Using this optimized algorithm, processes are no longer forced to manipulate lists of messages each time there is an i-view change, which makes the i-views a "light-weight" version of regular views. By reducing the overhead on the i-view changes, we reduce the impact of wrong suspicions due to aggressive failure detectors. Similarly, the fact that i-views do not force a process to suicide reduces the overhead induced by the membership service.

4.2 *k*-resiliency and the delivery latency

When evaluating the performance of a Total Order Broadcast, two important metrics are the *throughput* and the *delivery latency*. The *throughput* represents the rate at which the protocol can handle messages. Because *moving sequencer* algorithms are well known for their throughput performance [3, 6], we will not develop this aspect further.

On the other hand, the *delivery latency* may represent a problem on large networks [7]. The *delivery latency* represents the time elapsed between the broadcast of a message m by the source process and the first time it is delivered to the application. On fault-tolerant protocols, latency depends on the k-resiliency level, where k represents the number of processes that hold a copy of the message. Token-based protocol ensure resiliency by forcing process to acquire all previous messages before "accepting" the token. Indeed, when the sequence number s is sent by sequencer r, this implies:

- receiver r has all messages up to and including the s^{th} sequenced message,
- receiver (r-1) mod n has all messages up to and including the (s-1)th sequenced message,
- ...

Since total-resiliency may induce a high delivery latency on a large network, we may define a sufficient resiliency level that allows fault tolerance while providing a reduced latency. To better evaluate the impact of a k-resiliency over the delivery latency, we conducted preliminary experiments on a homogeneous cluster with 64 machines interconnected by a Fast Ethernet network. We measured the average delivery latency of 1kB messages using different k-resiliency levels. Fig. 4 presents the protocol performance when messages arrive at the rate of 10 messages per second in accordance with a Poisson process.

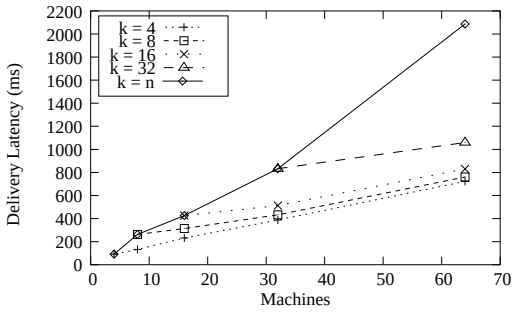

Figure 4: Delivery latency and *k*-resiliency

As expected, by limiting the k-resiliency we can bound the delivery latency. Similarly, the overhead observed when augmenting the number of processes gives some hints concerning the workload share. As there is a clear trade-off between k-resiliency and workload, distributed systems designers must be aware of its implications when developing fault tolerant applications.

5. RELATED WORK

In the literature there are some examples of protocols that rely on the *moving sequencer* strategy. RBP [3] first introduced the concept of *moving sequencer* total order broadcast. It was followed by RMP [15], which uses IP multicast to reduce the cost of a communication step. Furthermore, RMP includes some other features such as the acknowledgment of multiple messages in a single ack and the selection of the resiliency level. Both RBP and RMP use a Three-Phase Commit (3PC) membership.

TRMP [14] is a time driven version of the RMP protocol, presented as an Internet multicast protocol for the stock market. Due to the complexity of a world-wide distributed stock market system, TRMP has to deal with other problems like scalability, fairness and communication authentication, in addition to time constraints. To deal with scalability, TRMP proposes the use of a hierarchy of rings. This strategy minimizes the stabilization time, reduces the probability of failures in a *World-wide Reformation*, and allows the assignment of different levels of trustworthiness to the processes. These hierarchical rings are organized such that processes of the higher ring are also members of lower rings, relaying the messages. However, TRMP implements a centralized reformation server, with redundant reformation servers available in case of failures.

Ekwall and Schiper [8] proposed a different token-based total order broadcast relying exclusively on failure detectors instead of group membership. While this algorithm seems simpler and light-weight for homogeneous networks, it may lack accuracy in the case of complex dynamic environments. In [7] these authors compare different total order broadcast algorithms on wide-area networks, but the experimental scenarios give no further insight concerning pervasive systems.

Finally, Guerraoui [10] proposed FSR, an algorithm that is intended to optimize the throughput of a Total Order Broadcast algorithm. FSR is a hybrid approach based on the fixed-sequencer strategy that uses a token ring to ensure fairness among the nodes. Unfortunately, FSR uses a one-level membership, which forces the delivery latency to be linear function of the number of processes. We strongly believe that FSR can benefit from the two-level membership we propose to improve its latency and scalability.

6. CONCLUSIONS

In this paper we addressed the problem of Total Order Broadcast in the context of pervasive distributed systems. Traditional algorithms are not fit for these environments as they cannot handle the nodes volatility efficiently. We propose a self-stabilizing distributed solution that can operate in environments subjected to frequent disconnections while providing good performance rates. In order to ensure a smooth operation in spite of the volatility of the resources, we employed a new membership technique to minimize the problems generated by wrong failure suspicions, thus reducing the membership overhead and the need for view changes. To the best of our knowledge, this technique has not yet been used in any other implementation. Our efforts now concentrate on conducting experiments with more realistic environments, evaluating the impact of both nodes heterogeneity and volatility on the algorithm behavior.

7. REFERENCES

[1] D. Bottazzi, A. Corradi, and R. Montanari. Agape: a location-aware group membership middleware for pervasive computing environments. In *ISCC*, pages 1185–1192, 2003.

[2] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[3] J.-M. Chang and N. Maxemchuk. Reliable broadcast protocols. *ACM Trans. on Computer Systems*, 2(3):251–273, 1984.

[4] B. Charron-Bost, X. Défago, and A. Schiper. Broadcasting messages in fault-tolerant distributed systems: the benefit of handling input-triggered and output-triggered suspicions differently. In *Proceedings of the 21th International Symposium on Reliable Distributed Systems*, 2002.

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

[6] X. Défago. *Agreement-related Problems: from semi-passive replication to totally ordered broadcasts*. PhD thesis, EPFL, Switzerland, 2000.

[7] R. Ekwall and A. Schiper. Modeling and validating the performance of atomic broadcast algorithms. In *Proceedings of the 13th International EuroPar Conference*, LNCS Vol. 4641, pages 574–586, Rennes, France, Aug. 2007.

[8] R. Ekwall, A. Schiper, and P. Urbán. Token-based atomic broadcast using unreliable failure detectors. In *Proceedings of the 23rd Symposium on Reliable Distributed Systems (SRDS 2004)*, Florianópolis, Brazil, Oct. 2004.

[9] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[10] R. Guerraoui, R. Levy, B. Pochon, and V. Quéma. High throughput uniform total order broadcast protocol for cluster environments. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN)*, 2006.

[11] R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer, 2006.

[12] V. Hadzilacos and S. Toueg. *Fault-tolerant broadcasts and related problems*, chapter 5, pages 97–146. ACM Press Books, Addison-Wesley, second edition, 1993.

[13] L. Lamport. The part-time parliament. *ACM Transactions in Computer Systems*, 16(2):133–169, 1998.

[14] N. Maxemchuk and D. Shur. An internet multicast system for the stock market. *ACM Transactions on Computer Systems*, 19(3):384–412, 2001.

[15] T. Montgomery. *Design, Implementation and Verification of the Reliable Multicast Protocol*. PhD thesis, West Virginia University, 1994.

[16] H. Skaf-Molli, C.-L. Ignat, C. Rahhal, and P. Molli. New work modes for collaborative writing. In *International Conference on Enterprise Information Systems and Web Technologies (EISWT-07)*, Orlando, Florida, USA, July 2007.