Enforcing Strong Consistency with Semantically View Synchronous Multicast

José Pereira
Luís Rodrigues
Rui Oliveira

DI–FCUL TR–01–2

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/biblioteca/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Enforcing Strong Consistency with Semantically View Synchronous Multicast

José PEREIRA
Universidade do Minho
jop@di.uminho.pt

Luís RODRIGUES
Universidade de Lisboa
ler@di.fc.ul.pt

Rui OLIVEIRA
Universidade do Minho
rco@di.uminho.pt

Abstract

Replication is a fundamental strategy to obtain highly available services. Among the mechanisms that support replication, view synchronous multicast protocols emerge as a powerful abstraction to encapsulate fundamental problems in replication. Unfortunately, in the presence of a temporarily slow processor or network link the performance of implementations of view synchronous multicast is degraded for the whole group. This happens due to the strong reliability criterion, which forces a potentially large number of messages to be stored, eventually leading to buffer exhaustion and intermittent blocking of the application.

This paper proposes a new multicast primitive, Semantically View Synchronous Multicast, that alleviates this problem by selectively weakening reliability constraints while, at the same time, allowing strong consistency to be enforced at a higher level. The usefulness and practical relevance of the new primitive is illustrated using a modified primary-backup replication protocol.

1 Introduction

Group communication [14] is an elegant abstraction to encapsulate fundamental problems such as reliable communication and membership. It is therefore possible to implement fault tolerant highly available services using the primitives provided by group communication toolkits. For instance, the reliable and view synchronous multicast [7, 19] services offered by such toolkits have been shown to be an adequate foundation for primary-backup replication [6].

On the other hand, in practice the use of group communication in the implementation of services that require stable high throughput is impaired by temporarily slow replicas or network links. In fact, due to flow control mechanisms a single slow component affects the overall performance of the complete group [1]. This happens because the protocol may be forced to buffer a large number of messages, eventually leading to buffer exhaustion and to subsequent temporary blocking of the application. This is a fundamental problem inherent to strong reliability.

There are two main approaches to circumvent this problem. One is to exclude the slow member from the group [13, 5]. Although compatible with the usage of a group communication protocol, this forces the execution of an expensive reconfiguration procedure to re-establish the desired replication degree. The other approach consists in using a communication protocol with a relaxed reliability criterion thus accepting that some messages are lost [2]. However, if strict reliability is lost most of the simplicity obtained at the application level is also lost.

We address this with the proposal of a new primitive, called Semantically View Synchronous Multicast. This builds on our previous work on a relaxed reliability criterion based on notion of message obsolescence and on the subsequent definition of Semantically Reliable Multicast [11], which has been used in applications without inter-replica consistency requirements such as information dissemination systems.

The paper focuses on showing that despite the relaxed reliability criterion, this primitive is suitable to be used in a (strongly consistent) primary-backup replication protocol. Thereby, we present an algorithm for primary-backup replication together with a suitable definition of message obsolescence to exploit semantic reliability. Simulation results illustrating the practical relevance of the approach are also presented.

The paper is structured as follows: In Section 2 we present the assumed system model. In Section 3 we briefly introduce and discuss our definition of Semantic Reliabile Multicast. Section 4 introduces the primitive, Semantically View Synchronous Multicast and briefly addresses its implementation. Section 5 presents our primary-backup replication protocol and Section 6 evaluates its practical relevance. Section 7

concludes the paper.

2 System model

We consider an asynchronous message passing system augmented with an unreliable failure detector [4]. Briefly, we consider a set of sequential processes communicating through a fully connected network of point-to-point reliable channels. Asynchrony means that there are no bounds on processing or network delays. Processes can only fail by crashing and do not recover. A process that does not fail is considered *correct*. We assume that a majority of processes are correct and failure detection is of class $\diamond S$.

We assume the existence of a primary-partition group membership service [19] that manages the composition of a group of processes. The successive membership of the group is given by a sequence of *views* provided to the processes through view installation events. Processes may explicitly leave and rejoin the group.

3 Semantically Reliable Multicast

For self-containment and motivation we briefly introduce the notion of Semantically Reliable Multicast (SRM). A more thorough explanation of the problems addressed, the issues in its implementation and the resulting impact in performance can be found in [11].

3.1 Motivation

In a group, not delivering all messages to processes that are significantly slower than the majority makes it is possible to accommodate these processes without impairing the group's global performance [2]. However, if messages are arbitrarily dropped, regardless of the application being notified of the fact, most of the advantages of using a reliable multicast primitive are lost. Even when the application tolerates message loss, the code would become entangled by corrective measures to compensate the loss of messages.

Instead of dropping messages at random, we exploit the application's semantics to selectively drop messages that were made obsolete by subsequent operations. Our approach is based on the observation that in a distributed application messages often

overwrite or implicitly convey the content of other messages sent shortly before, therefore making them irrelevant. If messages become obsolete before their delivery, then they can be safely purged without compromising the application's correctness.

As an example, applying semantic reliability for information dissemination in a stock exchange application results in up to 40% additional delay at a receiver to be tolerated without impacting the performance of the group [11].

3.2 Definition

The definition of semantic reliability is based on obsolescence information formalized as a relation on messages. This relation is defined by the application program and encapsulates all the semantics ever required by the protocol. This way the SRM protocol can be developed independently of concrete applications.

For each pair of related messages $m \sqsubset m'$, we say that m is *obsoleted* by m'. The obsolescence relation is an irreflexive partial order (*i.e.*, anti-symmetric and transitive) and coherent with the causal ordering of events. The intuitive meaning of this relation is that if $m \sqsubset m'$ and m' is delivered, the correctness of the application is not affected by omitting the delivery of m. In this paper we consider only obsolescence among messages originating from the same sender. SRM is defined by the following properties:

Semantic Agreement: If a correct process p multicasts a message m and does not multicast a message m' such that $m \sqsubset m'$, then all correct processes deliver m.

Semantic Integrity: For every message m, every process p delivers m (i) at most once; (ii) never after delivering some message m' such that $m \sqsubset m'$ and (iii) only if m was previously multicast by some process.

A subtle issue that is worth to note in the above specification is, in runs where an infinite sequence of messages $m_1, m_2, \ldots$ such that for all i, $m_i \sqsubset m_{i+1}$ exists, the possibility of no message in the sequence to be ever delivered. This simplifies implementations and is not a problem for applications, as if an unbounded sequence of related messages exists and the consequent lack of delivery leads to blocking, any implementation will react by delivering some messages thus restoring liveness. Even if blocking does not happen, then the application can easily be modified to ensure that no such infinite sequences exist.

SRM offers however weak atomicity guarantees. In the agreement property of SRM nothing is ensured regarding message delivery when the sender fails: A process delivering a message m cannot tell whether the other processes are also going to deliver m

or some other message m' that makes m obsolete. While this is not a problem to certain applications [11] it is insufficient when strong consistency among a group of processes must be preserved.

4 Semantically View Synchronous Multicast

In this section we present the definition of the Semantically View Synchronous Multicast (SVSM) and briefly discuss how it can be implemented on top of SRM and a group membership service.

4.1 Definition

We propose a new primitive, Semantically View Synchronous Multicast, that combines the benefits of view synchronous communication with the advantages of semantic reliability. SVSM is stronger than SRM as it offers meaningful atomicity guarantees upon view installation events (which may be seen as synchronization points). Due to the obsolescence relation this does not mean that all processes deliver the same set of messages. It means that for a multicast set of messages all processes in the view deliver the maximal elements of this set regarding the obsolescence relation. Semantically View Synchronous Multicast is defined as follows:

Semantic View Synchrony: If a process p belonging to two consecutive views v_i and v_{i+1} delivers m in view v_i, then all processes in view v_i deliver m' before installing view v_{i+1}, such that $m' = m$ or $m \sqsubset m'$.

Notice that this property relaxes View Synchrony [15], as every pair of processes installing two consecutive views will not necessarily deliver the same set of messages but (at least) its maximal elements. Indeed, if no pair $m, m' \in M$ exist such that $m \sqsubset m'$, Semantic View Synchrony reduces to conventional View Synchrony. This makes Semantic View Synchrony a generalization of conventional View Synchrony, with an additional possibility of configuration through the obsolescence relation. As a consequence, algorithms that rely on the equivalence of state of processes upon view change will be easily adapted as it is illustrated in Section 5 with an example.

For this example we additionally assume FIFO Order and Sending View Delivery [19]. Briefly, FIFO Order states that no pair of messages originating from the same sender are delivered in the inverse order that they were multicast. Sending View Delivery states that messages are not delivered in a different view that they were multicast.

4.2 Implementation

Ignoring the obsolescence relation, the specification would be trivially satisfied by an implementation of View Synchronous Multicast. Such implementation would not be very useful, as it would not achieve the desired performance advantages. Although it is out of the scope of this paper to present a full implementation, we briefly discuss how it can be done.

An implementation can be obtained by observing that obsolescence is a stable property that can be recognized locally by a process which stores a pair of related messages. Using this knowledge, an existing protocol for View Synchronous Multicast can be modified in order to enforce view synchrony only for messages that are not known to be obsolete.

Consider for instance the implementation of view synchrony on top of a group membership service [16]. This protocol works by (i) using a reliable multicast primitive with an weak agreement property (*i.e.*, depending on the sender being correct) for message dissemination and then upon reception of a view (ii) by identifying the set of processes installing both the current and the next view and (iii) delaying view delivery until every message delivered by any process in that set is stable, *i.e.*, has been delivered by all processes in the set.

An implementation of SVSM can be accomplished similarly, although using SRM for message dissemination and then by removing obsolete messages from the set of messages that are required to be stable prior to view installation. In addition, the protocol used to achieve and detect stability can itself take advantage of semantic reliability given a suitable definition of the obsolescence relation.

5 Replication using SVSM

In this section we illustrate the usefulness of semantic reliability in strong consistent replication by presenting a primary-backup replication protocol that takes advantage of message obsolescence.

Primary-backup replication [3] works as follows: Upon receiving a *request* from a client, the primary server executes it, thereby modifying its state. Before replying to the client, it multicasts a *state-update* message for all replicas. Upon receiving an *acknowledgment* message from each of the replicas, a *reply* is sent to the client.

The use of SVSM to support a primary-backup protocol is interesting because in primary-backup replication state-update messages should be immediately sent to backup replicas in order to minimize the time required to send replies back to clients.

At that point, messages become out of reach of the replication protocol and cannot be discarded even if shortly after become obsolete. The use of a semantically reliable protocol, which allows messages to be discarded from protocol buffers, offers an elegant solution to the problem of balancing responsiveness requirements with memory constraints.

5.1 Primary-backup replication

For simplicity, we assume a fixed set of clients which issue requests sequentially. It is also not shown how the primary is selected. In addition, the recovery protocol required to let processes join the group is not described in detail: It is assumed that every process joining a view has upon installation the correct state obtained from some process present in both views. We also assume that execution of each upon statement in the algorithm is atomic.

To perform an invocation, a client sends the request to the process currently believed to be the primary. Until a reply is received, the request can be retransmitted if the client suspects the primary has failed. In order to distinguish retransmissions, each request message includes a sequence number which, along with the originator process identifier, can be used as a unique identifier for the request.

Handling of client requests constitutes the server protocol depicted in Figure 1. Only the current primary handles messages from clients (line 1) and only backups handle state update messages from the primary (lines 14 and 16). For clarity, the algorithm does not depict events that should be ignored, such as the reception of a client request by a backup server.

Servers keep some information about the last request processed for each client in $R[p]$ and $s[p]$ in order to handle duplicate requests which may be caused by retransmissions and that are necessary to cope with failover. Upon reception of a duplicate request a retransmission of the reply is performed (line 3). Otherwise, the request is executed (line 5) obtaining a reply y and a state difference D used to update backup replicas (lines 7 to 9). After acknowledgment messages have been received from all backup replicas, as counted by $c[p]$, the request is completed by sending the reply to the respective client (line 13).

5.2 Optimizing for message obsolescence

The state of the service is assumed to be a set of items that can be read and written while executing a request. As such, the difference between two states is a set D of pairs, each

Initialization:

$U \leftarrow <>$; $\forall p \in$ Clients, $R[p] \leftarrow \perp$, $s[p] \leftarrow 0$, $c[p] \leftarrow 0$

In the primary server:

```
1:  upon RECEIVE(req(i, q), p) do
2:      if s[p] = i then
3:          SEND(rep(i, R[p]), p);
4:      else if s[p] < i then
5:          (y, D) ← execute (q);
6:          R[p] ← y; s[p] ← i; c[p] ← 0;
7:          for each (d, u) ∈ D do
8:              SVSMCAST(upd(d, u));
9:          SVSMCAST(fin(i, y, p))

10: upon RECEIVE(ack(i, p), −) do
11:     c[p] ← c[p] + 1;
12:     if c[p] = |Servers| − 1 then
13:         SEND(rep(i, R[p]), p)
```

In each backup server:

```
14: upon SVSDELIVER(upd(d, u)) do
15:     U ← append (U, (d, u))

16: upon SVSDELIVER(fin(i, y, p)) do
17:     for each u of U do
18:         apply (S);
19:     U ← <>;
20:     R[p] ← y; s[p] ← i;
21:     SEND(ack(i, p), primary)
```

Figure 1: Primary-backup replication protocol.

constituted by a data item identifier d and an updated value u and is produced during the execution (line 5). When applied at the replicas (line 18), each update (d, u) sets the value of item d to u. Therefore, performing an update of an item overwrites its previous value.

Updating backup replicas is usually done by sending the state difference D in a single logical message. However, when considering semantic reliability this means the resulting message only becomes obsolete if a later update message refers to all the same data items, which is unlikely in practice.

To avoid this in the algorithm of Figure 1, updates to data items are sent as individual messages (in lines 7 and 8), followed by a finalization message containing the reply (in line 9). Atomicity is ensured by gathering updates (line 15) and applying them only when the finalization message is delivered (line 16).

Upon a view installation, queued updates in backup replicas for which no finalization message has been delivered are discarded. The primary fakes acknowledgment

messages from backup replicas that leave the group by appropriately incrementing $c[p]$ of pending requests.

As semantic reliability is being used, the primary-backup protocol is only complete after the obsolescence relation for messages has been defined. Two distinct types of messages are multicast (lines 8 and 9):

- update messages $upd(d, u)$, referring to a data item d;
- finalization messages $\mathit{fin}(i, y, p)$, containing an invocation identification (i, p) and a reply y.

Both are multicast by the primary after executing a request. This means that in addition to their contents, data available at that time, such as the complete set of updates D, can be used in the definition of the relation.

A naive definition that allows obsolete updates to be purged would be: $m \sqsubset m'$ iff $m = upd(d, u)$, $m' = upd(d', u')$, m precedes m', and $d = d'$. This definition would lead to inconsistency because it might violate atomicity of updates. As an example, consider a request q that updates two items, producing two update messages $upd(d, u)$ and $upd(d', u')$. A second request q' updates the same item d, resulting in a message $upd(d, u'')$. It would be possible for a backup replica to:

- deliver $upd(d', u')$ but not $upd(d, u)$ which is made obsolete by $upd(d, u'')$;
- deliver a finalization message for q and apply u';
- deliver $upd(d, u'')$;
- discard (d, u'') due to the installation of a new view.

Besides not having fully applied an update, this replica might be inconsistent with other replicas which have not discarded $upd(d, u)$, as allowed by semantic reliability.

This can be avoided with the definition of the obsolescence relation shown in Figure 2. This ensures that an update is only purged if a complete update set that makes it obsolete is delivered. In the previous example, the delivery of $upd(d, u)$ would only be omitted by the protocol if the finalization of q' is delivered, thus applying (d, u'') which makes (d, u) obsolete.

Given two messages m, m', then $m \sqsubset m'$ iff there are requests with associated updates q, D and q', D' such that:

- $m = upd(d, u)$ with $(d, u) \in D$;
- $m' = fin(y', i', p')$;
- the execution of q precedes the execution of q';
- $(d', u') \in D'$ exists such that $d = d'$.

Figure 2: Obsolescence relation.

5.3 Discussion

The complete primary-backup algorithm and its proof of correctness can be found in the extended version of this paper [12]. However, it is worth to highlight here the impact of the Semantic View Synchrony property in the correctness of the replication algorithm. For this purpose, allow us to assume the correctness of the algorithm of Figure 1 if based on View Synchronous Multicast.[1] Therefore, when considering Semantic View Synchronous Multicast the crucial issue is to show that upon a view change from v_i to v_{i+1} all replicas in the new view have exactly the same state, as this is the occasion when a new primary can be elected.

In view v_i, consider the set M_i of messages multicast by the primary of v_i to the backups. By (i) the Same View Delivery property of SVSM, in view v_i only update messages from M_i are delivered, and by (ii) the Semantic View Synchrony property of SVSM all replicas that install v_{i+1} deliver the very same set of maximal elements of M_i, and in the same order due to the assumption of FIFO channels. Since these maximal elements are the messages with guaranteed delivery and also those that allow the purging of other messages, the guarantee of replica consistency derives directly from the definition of the obsolescence relation.

With the relation defined in Figure 2 in mind, consider for contradiction, an obsolete update $upd(d, u) \in M_i$ (of item d with value u) that is not delivered at all replicas and therefore yields inconsistency in the value of item d. If $upd(d, u)$ is purged then, by the obsolescence relation (Figure 2) exists in M_i a maximal element $fin(y', i', p')'$ such that $upd(d, u) \sqsubset fin(y', i', p')'$ preceded by a maximal element $upd(d, u')$ (by the

[1] Except for the way our algorithm sends state updates to the backups, it is identical to that in [6]

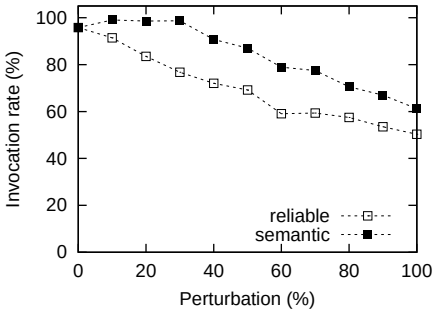

Figure 3: Impact of an increasingly slower replica on the performance of primary-backup replication.

algorithm and FIFO order) that overwrites $upd(d, u)$. Since $upd(d, u')$ is a maximal element of M_i it is delivered at all replicas installing v_{i+1} thus leaving d with value u' at all replicas. A contradiction of the hypothesis.

6 Practical relevance

To assess the practical relevance of our approach, we use an event-based simulation of the primary-backup algorithm using both strict and semantic reliability.

Using strict reliability, as implementations must rely only on a finite amount of buffers, if a member of the group is slow two alternatives are possible: (i) Client requests are temporarily denied through flow-control mechanisms; (ii) The slow member, recognized when the primary is idle waiting for replicas, has to be eventually excluded from the group in order to free buffers. Both approaches affect the availability of the replicated service. The first because it blocks the traffic (in practice preventing the service from being delivered), the second because it reduces the number of active replicas and thus, the service's resilience to process failures.

This is illustrated by the empty squares in Figure 3, which show throughput degradation when one replica is increasingly perturbed. For instance, if a maximum 10% performance degradation is accepted, a member has to be expelled approximately when 10% perturbed. These results were obtained using a set of 5 servers (one primary and 4 backups) and 10 clients. Upon execution, each request updates 1 to 5 (uniformly distributed) data items. Initially, the time taken to execute a request is the same as required to transmit and apply the resulting updates. This means that in stationary state the pri-

mary is fully used. The resulting throughput is considered to be the nominal capacity of the system, denoted by 100% in the results.

Using semantic reliability, it is relevant which is the access pattern to data items, which determines message obsolescence. For this we have used a real traffic pattern [11] in which some items are accessed more frequently than others. Skewed access patterns such as this are usually observed in on-line transaction processing systems, being an important feature of database benchmarking [18]. Buffering available at the primary is set to 5% of the amount required to hold all data items. Larger buffer sizer allow for more messages to be purged and thus larger delays to be tolerated.

In this situation, perturbing one replica has no impact in the overall system throughput while enough messages can be purged (*i.e.*, up to 30% perturbation), as is illustrated by filled squares in Figure 3. Using the same criterion as before, the system can now accommodate up to a 40% perturbation prior to expelling the slow replica. Notice that this happens despite acknowledgments being collected from all replicas for every request.

This has interesting implications for applications requiring stable high throughput and have a skewed data access pattern: Buffer space can be dimensioned in order to accommodate non-obsolete messages with a high probability and thus require expelling group members with very low probability even if a member of the group is significantly slower than the rest. This makes semantic reliability useful in situations where the time taken to execute a request at the primary is of the same magnitude or smaller that the time taken to transfer and apply updates to backup replicas. This is either because a backup replica is abnormally slow or behind a slow network link, or because the time taken to process the request is similar to the time taken to apply the update, as happens in I/O bound applications.

7 Conclusions

In this paper we have proposed the use of message semantics to improve throughput stability in reliable multicast protocols for applications with strong consistency requirements. This is done by selectively relaxing reliability based on a message obsolescence relation dictated by the application. Although for different purposes, application semantics has been used before to optimize group communication protocols. For instance, to relax causal order [8], total order [9, 10] and ordering of message deliveries with view changes [17].

We have presented the definition of a Semantically View Synchronous Multicast

primitive and have shown how it can be used in strongly consistent replication using a modified primary-backup protocol. The practical relevance of this is illustrated by showing that enforcing consistency does not prevent the achievement of the same performance advantages previously observed for applications without such requirements.

References

[1] K. Birman. A review of experiences with reliable multicast. *Software Practice and Experience*, 29(9):741–774, July 1999.

[2] K. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal multicast. *ACM Transactions on Computer Systems*, 17(2):41–88, 1999.

[3] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. The primary-backup approach. In Sape Mullender, editor, *Distributed Systems*, chapter 8, pages 199–216. Addison Wesley, second edition, 1993.

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

[5] B. Charron-Bost, X. Defago, and A. Schiper. Time vs space in fault-tolerant distributed systems. In *Proceedings of the Sixth International Workshop on Object-Oriented Dependable Systems*, Rome, Italy, January 2001.

[6] R. Guerraoui and A. Schiper. Fault-tolerance by replication in distributed systems. In *Reliable Software Technologies - Ada-Europe'96*, LNCS 1088, pages 38–57. Springer-Verlag, June 1996.

[7] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR94-1425, Cornell University, Computer Science Department, May 1994.

[8] R. Ladin, B. Liskov, and L. Shrira. Lazy replication: Exploiting the semantics of distributed services. *ACM SIGOPS Operating Systems Review*, 25(1):49–54, January 1991.

[9] S. Mishra, L. Peterson, and R. Schlichting. Consul: A communication substrate for fault-tolerant distributed programs. *Distributed Systems Engineering*, 1(2):87–103, December 1993.

[10] F. Pedone and A. Schiper. Generic broadcast. In *Proceedings of the 13th International Symposium on Distributed Computing (DISC'99, formerly WDAG)*, September 1999.

[11] J. Pereira, L. Rodrigues, and R. Oliveira. Semantically reliable multicast protocols. In *Proceedings of the Nineteenth IEEE Symposium on Reliable Distributed Systems*, pages 60–69, October 2000.

[12] J. Pereira, L. Rodrigues, and R. Oliveira. Enforcing strong consistency with semantically view synchronous multicast (Extended version). Technical report, Universidade do Minho and Universidade de Lisboa, In preparation.

[13] R. Piantoni and C. Stancescu. Implementing the Swiss Exchange trading system. In *Digest of Papers, The 27th International Symposium on Fault-Tolerant Computing Systems*, pages 309–313, Seattle, WA, July 1997.

[14] David Powell. Group communication. *Communications of the ACM*, 39(4):50–53, April 1996.

[15] A. Schiper and A. Sandoz. Understanding the power of the virtually-synchronous model. In *Proceedings of the 5th European Workshop on Dependable Computing*, Lisbon, February 1993.

[16] A. Schiper and A. Sandoz. Uniform reliable multicast in a virtually synchronous environment. In *Proceedings of the 13th International Conference on Distributed Computing Systems (ICDCS-13)*, pages 561–568, Pittsburgh, Pennsylvania, USA, May 1993. IEEE Computer Society Press.

[17] J. Sussman, I. Keidar, and K. Marzullo. Optimistic virtual synchrony. In *Proceedings of the Nineteenth IEEE Symposium on Reliable Distributed Systems*, pages 42–51, October 2000.

[18] Transaction Processing Performance Council. *TPC Benchmark C*. Shanley Public Relations, 777 N. First Street, Suite 600, San Jose, CA 95112-6311, May 1991.

[19] R. Vitenberg, I. Keidar, G. Chockler, and D. Dolev. Group communication specifications: A comprehensive study. Technical Report MIT-LCS-TR-790, The Hebrew University of Jerusalem and MIT, September 1999.