INTRUSION MASKING FOR DISTRIBUTED ATOMIC OPERATIONS

Meng Yu, Peng Liu, Wanyu Zang *

School of Information Science and Technology

Pennsylvania State University, University Park, PA 16803. USA.

{myu, pliu, wzang}@ist.psu.edu

Abstract Distributed database transactions are treated as atomic operations. This paper considers intrusion masking atomic operations among replica groups. The paper proposes two Byzantine fault tolerant two-phase commit protocols (BFT-2PC) to tolerate both malicious coordinator and malicious participants. When the total number of malicious replicas is no more than a specific threshold the protocols can mask attacks. Even if the number of faulty replicas is more than the threshold our protocols are still survivable in some cases. We analyzed the expected survivability and efficiency of our protocols. The analytic results demonstrate that our protocols are practical.

Keywords: Distributed systems, intrusion masking, survivability, atomic operations

1. Introduction

Intrusion tolerance is commonly achieved by replications [2, 6, 11, 14, 4] in distributed systems. Most of those techniques adopt the state machine approach [13] which ensures that replicated servers that start consistent will remain consistent as long as they apply the same deterministic actions in the same order. In the configuration of state machine approach a client sends requests to a group of replicated servers (a replica group) to obtain services.

State machine approaches can tolerate usually no more than one third malicious replica group members, a few lost or delayed messages, and those malicious clients that send inconsistent requests to different replica servers.

*We gratefully acknowledge funding support for this research. This research is supported by the Defense Advanced Research Projects Agency (DARPA) and Air Force Research Laboratory, Air Force Material Command, USAF, under agreement Number F30602-00-2-0575, by DARPA and AFRL, AFMC, USAF, under award number F20602-02-1-0216, by NSF CCR-TC-0233324, and by Department of Energy Early Career PI Award. This paper represents the opinions of the authors and does not necessarily represent the opinions or policies, either expressed or implied, of the United State government, DARPA, AFRL, AFMC, USAF, DOE, any of agencies.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: [10.1007/978-0-387-35691-4_52](https://doi.org/10.1007/978-0-387-35691-4_52)

D. Gritzalis et al. (eds.), *Security and Privacy in the Age of Uncertainty*

© IFIP International Federation for Information Processing 2003

At the same time, a hacked system may present arbitrary behaviors that can be modeled as Byzantine faults. So, Byzantine-fault tolerant algorithms can resist some malicious attacks. Applying Byzantine agreement algorithms to replicated database system for Byzantine fault tolerance has been discussed in [5], which suggests replicate both processing and data of whole data base system, namely full replication. Furthermore, the only necessary application of Byzantine agreement algorithm is submission of transactions to all replicas for resisting malicious attacks. This idea is essentially the same as the state machine approaches. Thus, the intrusion masking problem is often mistakenly considered easy to achieve by just using the total order services provided by group communication systems that are Byzantine-fault tolerant.

But both the state machine approach and others do not supply a directly solution of intrusion masking for distributed atomic operations, which are usually implemented by the two phase commit (2PC) protocol.

In the situation where neither coordinator nor participants are replicated, C. Mohan et al. proposed a Byzantine fault tolerant distributed transaction commit and recovery protocol [10]. They apply the Byzantine agreement algorithm in the second phase of 2PC to prevent the coordinator from sending inconsistent voting result to different participants. This protocol can only tolerate malicious coordinator in the second phase. But it obviously cannot tolerate malicious participants in the first phase of 2PC. Even a single malicious participant can paralyze the whole system by simply voting 'NO' to force the coordinator making a wrong 'abort' decision. Besides, the cost of Byzantine agreement algorithm is too expensive to be practical.

While more and more critical servers are replicated to replica groups the problem becomes more complex. For example, many banks could replicate their servers to replica groups to achieve high availability and survivability. There are already some qualified state machine approach based algorithms to guarantee the intrusion masking property when the client is a single host and makes a invocation to a replica group. The distributed transactions among banks are treated as atomic operations, which are either wholly committed or wholly aborted. When a replica group of one bank wants to have a transaction with other replica groups of other banks, we hope these distributed transactions are also intrusion masking. Therefore, we need new atomic commit protocols that can not only offer the capability of intrusion masking but also handle the replications.

The solution is not trivial. A normally functioned system should be able to abort distributed transactions due to resource contention, such as dead locks. We cannot simply grant a node as the delegate of a replica group to carry out the atomic commit protocol because such delegate nodes may be malicious and may work as we mentioned before to paralyze the system while we cannot distinguish it from good nodes.

The main contributions of this paper are as follows. As an example, we revised BFT [4] to demonstrate how to handle clients that are replica groups. Furthermore, we proposed two practical Byzantine-fault tolerant 2PC protocols for the distributed intrusion masking system. We analyzed the survivability and efficiency of these protocols.

2. Handling clients that are replica groups

2.1 Message Passing between Replica Groups

A replica group with n replicas is a n -size-group. Suppose there are m n -size-groups $\mathcal{G}_1, \mathcal{G}_2, \dots, \mathcal{G}_m$ in the system. For all $\mathcal{G}_i, 1 \leq i \leq m, |\mathcal{G}_i| = n$. We can construct *working group set* $\mathcal{D}$ according to the following rules. Each element d of $\mathcal{D}$ is a *working group*.

$$\begin{aligned} |\mathcal{D}| &= n \\ \forall d \in \mathcal{D}, |d| &= m \\ \forall d_i \in \mathcal{D}, d_j \in \mathcal{D}, i \neq j &\Rightarrow d_i \cap d_j = \phi, 1 \leq i, j \leq n \\ \forall a, b \in \mathcal{G}_i, a \in d &\Rightarrow b \notin d, 1 \leq i \leq m \\ \bigcup_{1 \leq i \leq n, d_i \in \mathcal{D}} d_i &= \bigcup_{1 \leq k \leq m} \mathcal{G}_k \end{aligned}$$

For all $\mathcal{G}_i, 1 \leq i \leq m$, we say $\mathcal{D}$ *contains* $\mathcal{G}_i$. Every replica in a n -size-group is assigned to a working group. Each element in a working group comes from different replica groups.

In Figure 2, $\mathcal{G}_1 = \{A_1, A_2, \dots, A_n\}$, $\mathcal{G}_2 = \{B_1, B_2, \dots, B_n\}$ and $\mathcal{G}_3 = \{C_1, C_2, \dots, C_n\}$ are three n -size-groups. We can construct $\mathcal{D} = \{\{A_1, B_1, C_1\}, \{A_2, B_2, C_2\}, \dots, \{A_n, B_n, C_n\}\}$. Each element in $\mathcal{D}$, $\{A_i, B_i, C_i\}, 1 \leq i \leq n$, is a working group. Please note the construction of set $\mathcal{D}$ is not unique according to our rules.

When a m -size-group $\mathcal{F}$ wants to send a request r to a n -size-group $\mathcal{G}$ there are two typical methods.

One method is *symmetric sending* when m is equal to n . A working group set $\mathcal{D}$ that contains $\mathcal{F}$ and $\mathcal{G}$ must be constructed. For all $a \in \mathcal{F}$, a sends $\langle r \rangle_{\sigma_a}$ to b if and only if $\exists d \in \mathcal{D}, a \in d, b \in d$ and $b \in \mathcal{G}$. The σ_a means r should be signed by a . Note the r in all messages should be the same because they are the same request and just sent by different replicas.

Another method is *asymmetric sending* whatever m is equal to n or not. For all $a \in \mathcal{F}$, a send $\langle r \rangle_{\sigma_a}$ to all replicas in $\mathcal{G}$. Then totally mn messages are sent. Each replica in $\mathcal{G}$ will receive m messages signed by different replicas in $\mathcal{F}$ and choose $f + 1$ consistent messages of them as the message they received provided $m = 3f + 1$. Like BFT we adopt $f + 1$ consistent messages as the weak certificate.

Two examples of symmetric sending and asymmetric sending are shown in Figure 1. Note that the two different methods have different efficiency and

Figure 1. Examples of symmetric sending and asymmetric sending

capability of fault tolerance. In the examples, replicas marked with checks are faulty replicas and messages marked with checks are faulty messages. The symmetric sending has less communication cost but worse capability of fault tolerance than those of asymmetric sending. A fault in a replica group will be propagated to the receiver in symmetric sending while asymmetric sending can resist such propagation of faults.

A distributed intrusion masking system consists of some replica groups, which can be organized in many different ways according to the method of message sending among them. We focus on two ways that we called *symmetric configuration* and *asymmetric configuration* in this paper.

In symmetric configuration a working group set must be constructed, which contains all replica groups in the system. Every two interactive replica groups in the system send messages by symmetric sending. An example of symmetric configuration is shown in Figure 2. The management of working groups, such

Figure 2. An example of symmetry configuration

Figure 3. An example of asymmetric configuration

as static assignment or dynamic management is beyond the scope of this paper.

In asymmetric configuration every two interactive replica groups in the system send messages by asymmetric sending. The number of replicas in each replica group may be different. An example of asymmetric configuration is shown in Figure 3. A_2 interacts with all replicas in other replica groups, such as $B_1, B_2, \dots, B_n$ and $C_1, C_2, \dots, C_k$. The asymmetric configuration does not

need management of working groups. So, asymmetric configuration is scalable and easy to be configured and managed.

2.2 Replica group as client

Based on BFT [4], we introduce a new Byzantine-Fault Tolerant protocol named BFT-S to meet the requirement of our communication schemes among replica groups. Please note our techniques are not limited on BFT. Our techniques can be applied to other similar protocols based on state machine approach.

We assume the client interacts with a $3k + 1$ -size-group. k is the maximum number of Byzantine-faulty replicas in the group that can be tolerated. In our architecture, either a single node or a replica group could be a client.

Phase request and reply. When the client is a $3k + 1$ -size-group and the system is under symmetric configuration the client symmetrically sends the $\langle \text{SYMMETRIC-REQUEST}, o, t, c \rangle_{\sigma_c}$ message to the $3k + 1$ -size-group. In the request t is the timestamp and o is the operation. c is a replica in the client. At the end of BFT-S each replica i in $3k + 1$ -size-group symmetrically sends reply $\langle \text{SYMMETRIC-REPLY}, v, t, c, i, r \rangle_{\sigma_i}$ to the client, where r is the result of request and v is the view number.

Otherwise, when the system is under asymmetric configuration (client is a single node or a replica group), or the client is a single node and the system is under symmetric configuration, the client asymmetrically sends the $\langle \text{ASYMMETRIC-REQUEST}, o, t, c \rangle_{\sigma_c}$ message to the $3k + 1$ -size-group. c is the client or a replica in the client. If the client is $3j + 1$ -size-group and the system is asymmetric configuration each replica in $3k + 1$ -size-group should wait $j + 1$ matching requests as the request. When the BFT-S finishes the reply $\langle \text{ASYMMETRIC-REPLY}, v, t, c, i, r \rangle_{\sigma_i}$ will be asymmetrically sent to the client. Client waits a weak certificate as the result, which is $k + 1$ matching replies that have the same v, t, r and valid signature as BFT.

Phase pre-prepare. The primary assigns a sequence number n to the request and multicasts the request to other replicas in the pre-prepare phase. Each replica will accept the request if and only if the request from client matches the request from primary. Otherwise, it will simply discard the request.

The prepare phase and commit phase are the same as BFT. They can be simply described as follows. In the prepare phase and commit phase, each replica broadcasts signed request to others and collects $2k + 1$ matching requests as a certificate to continue the following operations. For more information about

BFT please refer to [4]. A normal BFT-S procedure with single client is shown in Figure 4.

Figure 4. An normal BFT-S procedure

BFT-S improves BFT in three aspects. Firstly, a replica group can be a client of BFT-S. Secondly, in BFT-S all requests from the same replica group can be processed in parallel. This property will greatly reduce the unnecessary message cost in BFT-2PC protocols. Finally, clients have no necessary to know the primary of a replica group. We have the following theorem directly from BFT.

THEOREM 1 *BFT-S can tolerate no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty members of a n -size-group.*

3. Distributed Atomic Operations

Two Byzantine fault tolerant atomic commit protocols (BFT-ACP) are introduced in this section. Our BFT-ACP protocols can be combined with 2PC or 3PC protocols and named as BFT-2PC and BFT-3PC respectively. We discuss BFT-2PC under Symmetry Configuration (BFT-2PC-SC) and BFT-2PC under Asymmetry Configuration (BFT-2PC-AC) protocols in this section.

3.1 Two BFT-2PC Protocols

During the execution of a distributed transaction, the replica group that received the requests from clients is called *coordinator group* and the others are called *participant groups*.

After accepting a distributed transaction from a client, the coordinator group carries out the phase pre-prepare, prepare and commit of BFT-S to achieve an agreement. After that, the request will be logged into the stable storage for further recovery in case of malicious transactions. Then, BFT-2PC-SC or BFT-2PC-AC will be carried out according to the system configuration.

The BFT-2PC-SC protocol is described as follows.

Phase one: request to prepare. Coordinator group symmetrically sends $\langle \text{SYMMETRIC-REQUEST, REQUEST-TO-PREPARE, } t, c \rangle_{\sigma_c}$ to all partic-

ipant groups. Each participant group starts a BFT-S for the request to prepare.

Note t in the request should be same, which can be implemented by selecting the same one as that of the original distributed transaction. Because the t is used to guarantee the semantic of ‘exactly once’ so it will not be confused even if two different requests have same timestamps.

Phase two: voting on commit. Participant groups symmetrically send the reply $\langle \text{SYMMETRIC-REPLY}, v, t, c, i, \text{PREPARED} \rangle_{\sigma_i}$ or $\langle \text{SYMMETRIC-REPLY}, v, t, c, i, \text{NO} \rangle_{\sigma_i}$ to the coordinator group in terms of the transaction can be prepared or not.

Each replica in coordinator group makes a decision for ‘commit’ if all replies received from all participant groups are PREPARED. Otherwise, it decides ‘abort’. In both cases, all replicas should achieve agreement on their decisions to provide consistency, which is implemented by multicasting the decision of each replica and collecting $2f + 1$ matching decisions if the coordinator group is a $3f + 1$ -size-group.

The coordinator group notifies all participant groups to ‘commit’ or ‘abort’ the transaction by symmetric sending the request $\langle \text{SYMMETRIC-REQUEST}, \text{COMMIT}, t, c \rangle_{\sigma_c}$ or $\langle \text{SYMMETRIC-REQUEST}, \text{ABORT}, t, c \rangle_{\sigma_c}$ to all participant groups respectively. Participant groups start another BFT-S procedure for the notification. Finally participant groups symmetrically send the reply $\langle \text{SYMMETRIC-REPLY}, v, t, c, i, \text{DONE} \rangle_{\sigma_i}$ to the coordinator group.

After the coordinator group receives DONE, the coordinator group sends replies to the client.

The BFT-2PC-AC protocol can be exactly described as the same as BFT-2PC-SC by replacing all symmetric sending with asymmetric sending and replacing all SYMMETRIC-REQUEST, SYMMETRIC-REPLY with ASYMMETRIC-REQUEST and ASYMMETRIC-REPLY.

3.2 Survivability

In this paper, we define that the *survivability* of a system is the probability that the system can function correctly under a given condition.

A working group d is *correct* if and only if every replica in d is not faulty and the communication among them is not faulty. Otherwise, the working group is a *faulty working group* because the operation among them cannot obtain right results.

THEOREM 2 *Provided in a distributed transaction the coordinator group is n -size-group and interacts with other participant groups, then, BFT-2PC-SC can survive no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty working groups.*

PROOF SKETCH. Derived from Theorem 1. $\square$

Please note in the Theorem 2 the number of faults are quantified by the number of faulty working groups but not the number of faulty replicas in each replica group. This is because the faults can be propagated in working groups. As shown in Figure 1, before the symmetric sending, each 4-size-group have one faulty replica that all of them can survive. After the symmetric sending, the fault of sender was propagated to the receiver, which leads to the receiver cannot survive such faults.

COROLLARY 3 *Provided there are m n -size-groups in the system under symmetric configuration, if the probabilities that f faulty replicas occur at any place are equal, then, BFT-2PC-SC can survive at least k faulty replicas. If $k < f \leq mk$, the survivability P of system can be described with equation (1). BFT-2PC-SC cannot survive more than mk faulty replicas.*

$$P = \frac{\binom{n}{k} \binom{mk}{f}}{\binom{mn}{f}} \quad (1)$$

where $k = \lfloor \frac{n-1}{3} \rfloor$.

PROOF. Omitted. $\square$

THEOREM 4 *BFT-2PC-AC can survive no more than $\lfloor \frac{n-1}{3} \rfloor$ faulty replicas in each n -size-group in the system.*

PROOF SKETCH. Derived from Theorem 1. $\square$

COROLLARY 5 *Provided there are m n -size-groups in the system under asymmetric configuration, if the probabilities that f faulty replicas occur at any place are equal, then, BFT-2PC-AC can survive at least k faulty replicas. Beside this, If $k < f \leq mk$, the survivability P of system can be characterized with equation (2). BFT-2PC-AC cannot survive more than mk faulty replicas.*

$$P \geq \max \left(0, \frac{\binom{mn}{f} - m \sum_{i=k+1}^{\min(f,n)} \binom{n}{i} \binom{(m-1)n}{f-i}}{\binom{mn}{f}} \right) \quad (2)$$

where $k = \lfloor \frac{n-1}{3} \rfloor$.

PROOF. Omitted. $\square$

The Corollary 5 gives a lower bound of survivability of BFT-2PC-AC. When m is small enough (e.g. $m = 2$ or $m = 3$) the corollary gives the accurate survivability.

When there are m n -size-groups in the system, we compared the expected survivability of BFT-2PC-SC and BFT-2PC-AC. The expected survivability

Figure 5. Comparing the survivability of BFT-2PC-SC and BFT-2PC-AC

Figure 6. Time model

when m and n have different values is shown in Figure 5. We observed that BFT-2PC-AC always has better survivability than that of BFT-2PC-SC. We believe this is because the asymmetric sending has better capability of fault tolerance. The asymmetric structure of BFT-2PC-AC also contributes to its survivability.

3.3 Efficiency

The efficiency of protocols can be characterized in many aspects. We focus mainly on the communication cost and the latency in this paper.

The communication cost can be quantified by the number of messages, which depends on the implementation of the communication layer. For example, multicast could be implemented by unicast or broadcast in the lower layer. The communication costs of two methods are totally different. So we count for the messages for both two cases.

As for the latency of processing, we adopt the time model as shown in Figure 6, which is similar to BFT's.

In the figure U is the time of multicasting. All messages received by receiver are queued in the buffer of receiver. The receiver needs time TQ to process the message queue. Finally, the receiver needs time TE to execute the request.

Based on our time model, the latency of a procedure can be described as equation (3) in its critical path.

$$L = \sum U + \sum TE + \sum TQ \quad (3)$$

Because all participants can execute in parallel, so in the critical path, TE can be denoted by equation (4).

$$\sum TE = T_T + T_C + 2 \max T_P + T_V + t_\sigma N_{sig} \quad (4)$$

In the equation, T_T is the executing time of the transaction, T_C is the time of executing one BFT-S by coordinator group, T_P is the time of executing one BFT-S by participant group, T_V is the time of voting in coordinator group, t_σ is the signature time for a single message and N_{sig} is the total number of messages that need be signed in the critical path.

Regardless the topology of network and the detail of implementation, for simple analysis we just assume the multicast time U is in proportion to the total number of messages N and TQ is in proportion to the total length of message queue N_q in the critical path. Then we have equation (5).

$$\sum U + \sum TQ = t_m N + t_q N_q \quad (5)$$

In the equation t_m is the unit time to transmit a message and t_q is the unit time to process a message in the queue. So, we have the latency described with equation (6).

$$L = T_T + T_C + 2 \max T_P + T_V + t_\sigma \sum N_{sig} + t_m N + t_q N_q \quad (6)$$

Suppose we have m n -size-groups, they can be configured in either symmetry or asymmetry. We compared their communication costs and latency. The parameters for latency evaluation is shown in Table 1. Some default values

Table 1. The parameters for latency evaluation

Symbol Definitions	Default Values
T_T : executing time of a transaction	1.2 second
T_C : time of one BFT-S by coordinator	0.1 second
T_P : time of one BFT-S by participant	0.1 second
T_V : voting time of coordinator	0.04 second
t_σ : signature time of a single message	0.1 second
t_m : time for transmitting a message	0.02 second
t_q : time to process a message in the queue	0.005 second

about transactions and communication in the table are chosen based on [8]. When the multicast is implemented by unicast communication the latency of two BFT-2PC protocols is shown in Figure 7. According to the results, BFT-2PC-AC is not a pragmatic scheme due to its poor latency while BFT-2PC-SC demonstrates better results. Also the total number m of replica groups has great impact on the latency of BFT-2PC-AC.

When the multicast is implemented by broadcasting the latency of two BFT-2PC protocols is shown in Figure 8. In the figure, the latency of BFT-2PC-SC and BFT-2PC-AC do not deviate too much. Both of them have reasonable latencies and are practical.

Figure 7. Latency when multicast is implemented by unicasting

Figure 8. Latency when multicast is implemented by broadcasting

4. Discussion

Our two BFT-2PC protocols demonstrated how to carry out distributed operations among replica groups while offering the capability of intrusion masking. Other distributed operations, such as termination protocol, failure handling and optimization [3](e.g. Presumed Abort or Presumed Commit) can be constructed in similar approach.

Quorum replication [9] can also have the capability of intrusion masking. But quorum replications usually restrict operations (read and write) and needs locks to access variables. State machine approach based algorithms has no limitation on operations and could have better performance as BFT does.

Because our approaches bases on state machine approach they cannot handle consistent malicious requests sent by malicious clients. The techniques of recovery from malicious transactions [1] have been studied in recent years. These techniques can be integrated into our scheme because we can store accepted requests into stable storage. When the Intrusion Detection System (IDS) recognized the malicious requests, the system should isolate and confine the destroyed data, then start a consistent recovery request to recover from malicious transactions. If the malicious transactions are distributed, the recovery can be carried out with our BFT-2PC protocols.

Distributed operations implemented by reliable multicast and voting have been suggested in [12, 7]. Their communicating costs and voting cost are much more than ours.

5. Conclusions and Future Work

We proposed two intrusion masking 2PC protocols for distributed operations. The survivability and efficiency of two protocols have been studied. According

to our analytic results both of two BFT-2PC protocols can survive Byzantine faults and are practical. No Byzantine agreement algorithm is needed in our protocols. We will apply our techniques into other distributed operations and investigate the impact in the future researches.

References

- [1] P. Ammann, S. Jajodia, and P. Liu. Recovery from malicious transactions. *IEEE Transaction on Knowledge and Data Engineering*, 14(5):1167–1185, September/October 2002.
- [2] Anish Arora and Sandeep S. Kulkarni. Designing masking fault-tolerance via nonmasking fault-tolerance. *IEEE Transactions on Software Engineering*, 24(6):435–450, June 1998.
- [3] Philip A. Bernstein and Eric Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997. ISBN 1-55860-415-4.
- [4] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *The Third Symposium on Operating Systems Design and Implementation (OSDI '99)*, New Orleans, USA, February 1999.
- [5] Hector Garcia-Molina and Frank Pittelli. Applications of byzantine agreement in database systems. *ACM Transactions on Database Systems*, 11(1):27–47, March 1986.
- [6] Felix C. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1):1–26, March 1999.
- [7] Bettina Kemme, Fernando Pedone, Gustavo Alonso, and Andr Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proc. 19th IEEE International Conference on Distributed Computing Systems*, pages 424–431, Austin, TX, USA, 31 May - 4 June 1999.
- [8] Xiangning (Sean) Liu and Weimin Du. Multiview access protocols for large-scale replication. *ACM Transactions on Database Systems*, 23(2):158–198, June 1998.
- [9] Dahlia Malkhi and Michael Reiter. Byzantine quorum system. *Distributed Computing*, 11(4):203–213, November 1998.
- [10] C. Mohan, R. Strong, and S. Finkelstein. Method for distributed transaction commit and recovery using byzantine agreement within clusters of processors. In *Proceedings of the second annual ACM Symposium on principles of distributed computing*, 1983.
- [11] G. Morgan and P.D. Ezhilchelvan. Policies for using replica groups and their effectiveness over the internet. In *Proc. 2nd International COST264 Workshop on Networked Group Communication (NGC 2000)*, Palo Alto, California, 2000.
- [12] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Strong replica consistency for fault-tolerant corba applications. In *Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'01)*, pages 10–, Rome, Italy, January 08 - 10 2001.
- [13] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Computing Surveys*, 22(4):299–319, December 1990.
- [14] Ciprian Tutu Yair Amir. From total order to database replication. In *22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 494–, Vienna, Austria, 1992.