1. Practical Impact of Group Communication Theory

André Schiper

Ecole Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
andre.schiper@epfl.ch

1.1 Introduction

Group communication is nowadays a well established topic in distributed computing. It emerged over the years as a topic with a strong synergy between theory and practice: group communication is highly relevant for building distributed systems, and is also of theoretical importance, because of the difficult problems it addresses. The paper presents an – inevitably subjective – retrospective of the main milestones that led to our current understanding of group communication, with the focus on theoretical contributions of practical relevance. Some open issues are discussed at the end of the paper.

What is theory? In the context of group communication, theoretical contributions can be defined as contributions to *abstractions* and *paradigms*, contributions to *system models* and *problem specifications*, and of course contributions to *algorithms*.

What is of practical importance in the context of group communication? We can mention *efficiency*, *clean structure (or architecture) of the system*, *flexibility of the system*, *correctness*, and *clear understanding of the properties of the system*. Efficiency is clearly important: no one would like to have an inefficient group communication system. *Algorithms* and *paradigms* may contribute to efficiency: a new algorithm, or the application of a new paradigm, may increase efficiency. *Abstractions* and *algorithms* contribute to a clean and flexible architecture of the system. Consider for example atomic broadcast. The algorithm in [1.11] solves atomic broadcast by reduction to consensus. This identifies consensus as a key abstraction, and directly influences the architecture of a group communication system by introducing a *consensus* component. A system with an architecture that reflects key *algorithmic abstractions* has more chance to be flexible, i.e., adaptable to a changing environment. Correctness, which obviously is of practical importance, is related to *specifications* (what is the system supposed to do), to *system models* (what are the assumptions about the system) and to *algorithms* (which are proven to meet a given specification in a given system model). Clear understanding of the properties of the system can be seen as a subset of correctness. When a system is deployed, it is important to know the conditions under which the system can deliver the services it is supposed to provide. This requires understanding of the *specifications* of the system, of the *system models* and *algorithms*, and also of *theoretical results* (e.g., the FLP impossibility result [1.26], or the result about the weakest failure detector for solving consensus [1.10]).

To summarize, the retrospective addresses the practical impact of group communication theory by focusing on key contributions to *abstractions*, *algorithms*, *paradigms*,

A. Schiper et al. (Eds.): Future Directions in DC 2002, LNCS 2584, pp. 1–10, 2003.

specifications, *system models* and *theoretical results*. Contributions are grouped into three periods[1]:

- *Prehistory* (1972 - 1985)
- *Early years* (1985 - 1991)
- *Maturity and confusion* (1992 - 2002)

We start our retrospective in 1972, the year of the publication of the first paper that attempts to propose a completely software-based approach to fault-tolerance [1.44]. We end the "prehistorical" period in 1985 with the publication of the FLP impossibility result in the Journal of the ACM [1.26]. The year 1985 marks also the beginning of the "early years" period, with the publication of the first Isis paper [1.9]. We end this period in 1991, with the publication of the failure detectors paper [1.11], an important step in the history of group communication. The "maturity and confusion" period starts with the publication of the first partitionable group membership paper [1.3].

1.2 Prehistory (1972–1985)

We give only a brief overview of the main contributions the first period 1972-1985. In the context of *abstractions* and *specifications*, it is interesting to note that fundamental abstractions are identified: *interactive consistency* [1.36], *consensus* [1.25], *atomic broadcast* [1.13, 1.19]. The focus at that time was on Byzantine failures [1.45, 1.33], which shows that the complexity of solving problems with crash failures only was underestimated. The early focus on Byzantine failures had a major impact on problem specification, i.e., it led to consider *non-uniform* specifications, and the interactive consistency problem.

The *state machine paradigm* is also identified in the prehistorical period, and its implementation discussed in various system models: with no crashes [1.30], in the context of crash failures [1.42] and in the context of Byzantine failures [1.31]. The *rotating coordinator paradigm* is also already mentioned [1.40], even though it became well known only later (see Section 1.3.2).

Strong system *models* are considered in that period, e.g., the *synchronous round* model [1.36] and the *fail-stop* model [1.43], which corresponds to the asynchronous model with perfect failure detection. Models with randomization, for solving consensus, are also proposed [1.6, 1.39]. Finally, the FLP impossibility result [1.26], stating that there is no deterministic algorithm for solving consensus in an asynchronous system if one single process may crash, ends this initial period. The result will move the focus from Byzantine failures to crash failures.

1.3 Early Years (1985–1991)

1.3.1 Abstractions and Specifications

Process groups, *virtual synchrony* and *group membership* appear in the period 1985-1991. The notion of *process group* was proposed initially in an operation systems paper,

[1] The chronology of the published papers (conference or journal version) sometimes differs from the chronology of the corresponding technical reports. If the interested authors send me the historical TR references, I will add them in an extended version of this paper.

in the context of the V System [1.17]. In the paper, Cheriton and Zwaenepoel mention operations such as *join group*, *leave group*, *send message to group*, etc. It is interesting to notice that the paper also mentions the *publish-subscribe* paradigm. The concept of *virtual synchrony* was introduced in the paper by Birman and Joseph, as a specification that encompasses atomic broadcast (abcast), causal broadcast (cbcast) and group broadcast (gbcast) [1.8]. The paper does not give any precise specification of these group communication primitives, but stresses on the benefit of using these abstractions to develop fault-tolerant software:

> *We argue that this approach to building distributed and fault-tolerant software is more straightforward, more flexible and more likely to yield correct solutions than alternative approaches*.

The *group membership* abstraction and its specification appears in two papers in 1991: [1.41, 1.20]. The two papers give specifications for what is called today the *primary partition* group membership. [1.41] focusses on *processor* membership (i.e., the members of the group are processors) in an asynchronous system model, and advocates group membership as a mean to provide consistent failure notification. The specification was later shown to be flawed (see Section 1.4.2). [1.20] discusses *processor* and *process* membership in a synchronous system model. The process membership information is obtained from the processor membership. The goal of group membership is here to solve the continuous leader election problem.

1.3.2 Paradigms

We have already mentioned the *rotating coordinator* paradigm in the period 1972-1985. However, the paradigm became really known in the "early years" period, thanks to two papers [1.23, 1.11], which use the paradigm to solve consensus. In the rotating coordinator paradigm, the coordinator role moves from one process p to another process q, in a predetermined order, whenever p is suspected. Once all processes have been suspected, the coordinator role returns to the first coordinator, etc. A process can thus become the coordinator more than once. The paper by Chang and Maxemchuck [1.14], which uses a rotating token to implement atomic broadcast, is sometimes mentioned in the context of the rotating coordinator paradigm. However, token passing in [1.14] is not related to failure suspicions, and thus does not correspond to the definition.

1.3.3 System Models

The synchronous round model and the fail-stop model mentioned in Section 1.2 are very constraining from a practical point of view: they do not allow wrong failure suspicions. From a practical point of view one would like to have a model that allows the system to mistakenly suspect correct processes. Two such system models were proposed in that period: the *partially synchronous* model [1.23] and the *failure detector* model [1.11].

The partially synchronous model considers bounds on the message transmission delay and on the relative speed of processes. There are two variants of the model: (1) the bounds exist, but are not known, and (2) the bounds are known, but hold only from some unknown point on. The second variant is usually considered, but the first variant is actually more appealing from a practical point of view.

The failure detector model can be seen as a refinement of the partially synchronous model. A failure detector is defined by a *completeness* property and by an *accuracy* property. The completeness property defines the behaviour of the failure detectors with respect to faulty processes, i.e., processes that crash. The accuracy property defines the behavior of the failure detectors with respect to correct processes.

1.3.4 Algorithms

Many group communication algorithms were published in the period 1985-1991, and it is impossible to mention them all. In chronological order, the two first papers to mention are related to the group membership abstraction [1.21, 1.1]. The first paper defines *dynamic voting* (or *dynamic quorums*) [1.21], the second *view-based quorums* [1.1]. The two papers introduce some flavor of "primary partition group membership". In [1.21], each site maintains some information, which allows the site to determine who is in the same partition. In [1.1], the authors write: *"Each site s maintains a set called its view, the set of sites s assumes it can communicate with. Associated with each view is a view-id and two sites are said to be in the same view if their views have identical views-ids"*. Read and write quorums are based on views. The two papers show the algorithmic benefit of a dynamic membership information. This became less clear later . . . (see Section 1.4.2).

While the benefit of group communication abstractions is discussed by Birman and Joseph in [1.8], the same authors present the algorithms that implement these abstractions in [1.7]. The paper presents algorithms for causal broadcast (based on piggybacking on every message m the messages that are in the causal past of m) and for atomic broadcast, based on an idea by Skeen (for each message m a sequence number $sn(m)$ is computed, and messages are delivered in the order of their sequence number). The paper discusses also global broadcast (gbcast), with a rather complicated algorithm. Even though the paper lacks of rigorous specifications for group communication primitives, and does not address liveness, it constitutes a major milestone in the history of group communication.

Consensus algorithms are the key algorithmic contributions in the period 1985-1991: (1) the consensus algorithm for the partially synchronous model based on the rotating coordinator paradigm [1.23], (2) the Paxos algorithm based on leader election [1.32], and (3) consensus algorithms for the asynchronous system augmented with failure detectors [1.11], specifically the consensus algorithm using the $\diamond\mathcal{S}$ failure detector and based on the rotating coordinator paradigm. These three algorithms have in common the separation of correctness into safety and liveness properties, where the safety properties hold no matter how asynchronously the system behaves. Termination (i.e., liveness) requires some additional condition.

It is interesting to read Lamport's comments about Paxos [1.29]: the algorithm was submitted for publication in 1990, but not published before 1998. It was initially not understood, and the reviewers of found the paper *"mildly interesting, though not very important"*. . . Currently it is still an open question whether it is more efficient to solve consensus using a rotating coordinator or leader election.

Another key contribution is the atomic broadcast algorithm by Chandra and Toueg [1.11] based on reduction to consensus: the atomic broadcast algorithm inherits the properties of the consensus algorithm, e.g., safety holds no matter how asynchronous the

system is. The algorithm also shows the practical importance of consensus. References of other atomic broadcast algorithms can be found in [1.22].

1.3.5 Summary

To summarize, in the period 1985-1991, the notion of group membership appears (implicitly and explicitly), the partially synchronous and the failure detector system models are defined, and the rotating coordinator paradigm becomes popular. In terms of algorithms, the main contributions are consensus algorithms whose safety properties hold even if the system behaves completely asynchronously, and the atomic broadcast algorithm solved by reduction to consensus.

1.4 Maturity and Confusion (1992–2002)

The contributions from 1985 to 1991 establish the theoretical basis for group communication and lead to substantial progress in the field during the next ten years. New topics continue to emerge during this period, and some of these topics serve as a catalyst for further research on theoretical foundations.

1.4.1 Maturity

The elegance and the power of failure detector model have influenced a large number of researchers, and generated an important literature. Other important results were also established.

Abstractions, Specifications and System Models. In the context of abstractions and specifications, the paper by Hadzilacos and Toueg [1.27] has played an important role: it has contributed in part by motivating subsequent authors to pay more attention to a careful specification of group communication. A new group communication primitive, called *generic broadcast* [1.38], which takes the message semantics into account, was also proposed. The primitive is parametrized by a *conflict* relation, and ensures that two conflicting messages are delivered in the same order by all processes, while non-conflicting messages need not to be ordered.

In the context of system models, the failure detector model, initially defined in a model where processes do not recover after a crash, has been extended to a model with process recovery [1.2]. The paper introduces the notion of *good* process, a process that is always up or eventually permanently up, and of *bad* process, a process that is eventually permanently down or unstable (i.e., permanently crashing and recovering). The paper also introduces the failure detector $\diamond\mathcal{S}_u$, based on epoch numbers, for solving consensus in the crash-recovery model.

The increasing maturity of the field is also witnessed by the comprehensive survey on group communication specifications by Chockler, Keidar and Vitenberg [1.18]. The "consensus" survey by Barborak, Malek and Dahbura [1.5] may also be mentioned here, even though the paper is written from a *system diagnosis* perspective, rather than from a *distributed computing* point of view.

Theoretical Results. Two important theoretical results were established in this period. One, by Chandra, Hadzilacos and Toueg, shows that $\diamond\mathcal{S}$ (or equivalently $\diamond\mathcal{W}$) is the weakest failure detector that allows us to solve consensus in an asynchronous system.

This has a very practical importance: a system that uses a $\diamond\mathcal{S}$-based consensus algorithm, solves consensus in the maximum possible runs. Another result is the impossibility of solving the group membership problem in an asynchronous system [1.12]. Before this paper it was sometimes claimed that the group membership problem was not subject to the FLP impossibility result, because the model allowed processes to crash (and to kill other processes).

The following works are also worth mentioning. One is the paper by Malkhi and Reiter about *Byzantine* quorums, which significantly extended previous results and renewed the research on quorum systems, e.g., [1.34]. The other work is by Fekete, Lynch and Shvartsman [1.24] in which the authors formally specify and *prove* a view-oriented group communication service (the proofs were subsequently checked by a theorem prover).

Algorithms. Many algorithms were published in this period, e.g., consensus algorithms based on failure detectors[2] and algorithms for solving other agreement problems by reduction to consensus (atomic commitment, atomic multicast, group membership). The most important algorithm to mention is probably the algorithm for solving consensus in the static system model with process recovery, based on the failure detector $\diamond\mathcal{S}_u$ [1.2].

1.4.2 Confusion

Despite the maturity of the domain, group communication is far from being a solved problem. Many contributions, including theoretical contributions, are still needed in order to clarify some controversial issues, mostly in the context of the group membership problem. We mention first the CATOCS controversy (causally and totally ordered communication support), which generated lots of discussions some years ago.

The CATOCS Controversy. Cheriton and Skeen expressed several criticisms to the use of group communication to build distributed applications [1.16]. These criticisms, which have led to the so-called CATOCS controversy, can be summarized as follows: (1) group communication cannot guarantee total ordering between operations that correspond to groups of messages (e.g., transactions), (2) there is no efficiency gain over state-level techniques, (3) semantic ordering cannot be expressed, and (4) group communication does not ensure end-to-end guarantees. According to these criticisms, group communication has not brought anything useful to distributed computing. This is incorrect.

First, group communication define new abstractions, e.g. atomic broadcast. Abstractions are important, and the whole development of computer science consists of identifying new abstractions that contribute to the understanding of the field. Moreover, abstractions allow the development of more complex applications, and decrease the risk of errors. Do abstractions lead to inefficient solutions? This is an old story[3]. While early implementation of group communication where known to be slow, this is no longer

[2] It is interesting to note that already in 1983 (!), Fischer writes *"We survey the considerable literature on the consensus problem that was developed over the past few years and give an informal overview of the major theoretical results in this area"*. I don't know what should be said today . . .

[3] In the early seventies, some people were claiming that it will never be possible to write programs more efficiently than using assembly languages. I don't know whether these people would be eager to program a modern RISC processor at the assembler-language level!

the case with current LAN technology. For example with 100Mbits/s Ethernet, atomic broadcast becomes extremely fast, e.g., around 500-1000 atomic broadcasts per second.

It is true that group communication as such does not guarantee total ordering between operations that correspond to groups of messages. However, this does not prevent providing higher level ordering guarantees using group communication. A good example are the recent results showing the benefit of using atomic broadcast to implement transactions over a replicated database [1.37, 1.28, 1.46]: the use of atomic broadcast greatly simplifies the solution, while leading to better performances over standard database replication techniques. It is also false to claim that semantic ordering constraints cannot be expressed using group communications. Generic broadcast is an example of message ordering communication primitive that uses message semantics [1.38].

The comment about the absence of end-to-end guarantees available from group communication is maybe the most interesting. First, it is true that the absence of end-to-end guarantees can lead to problems. For example, it has been shown that the absence of end-to-end guarantees does not allow one to implement *2-safe* transactions over a replicated database using atomic broadcast [1.46]. However, (1) end-to-end guarantees can be added to group communication primitives, and (2) the absence of end-to-end guarantees can sometimes be an advantage. For example, it can be exploited to define a new safety property for transactions called *group-safety*, weaker than 2-safety, which can be sometimes more efficient than *lazy replication* while providing much stronger guarantees [1.46].

The Group Membership Issue. The group membership issue is of a different nature than the CATOCS controversy. Here, more contributions are definitely needed to clarify important issues.

The year 1992 corresponds to the publication of the first *partitionable* membership paper [1.3]. In that paper group membership is defined as *"maintaining the Current Configuration Set in consensus among the set of machines that are connected throughout the activation of the membership protocol"*. If this first definition leaves many questions open, ten years later the fundamental questions related to partitionable membership have still not been adequately addressed. For example, in [1.18] Chockler *et al.* give the following specification in the context of the partitionable membership problem: *"If the failure detector behaves like $\diamond\mathcal{P}$, then for every stable component S there exists a view $V = S$ such that every process in S installs V as its last view"*. This looks like the detection of a global (stable) property. Is the partitionable group membership problem an agreement problem, or does it rather correspond to the detection of a stable property?

While the specification of the partitionable membership problem is one issue, the role of this abstraction is still unclear. Convincing examples are needed. While the Isis *processor* membership abstraction [1.41] has clear limitations (one single group, all processors member of this group, progress only in the majority partition of the network), this limitation can be left by relying on multiple *process* groups, with one group for every set of replicated servers. Partitionable membership is not needed for solving agreement problems when the network can partition.

These comments may suggest that the simpler primary partition membership problem is fully understood. Unfortunately, this is not the case. Many specifications have been published, but none of them is satisfactory: it has been shown that current specifications

(partitionable, but also primary partition) admit trivial solutions or allow for undesirable behavior [1.4].

The question of the algorithmic role of the primary partition membership problem requires also some clarification. One role was clearly identified about 15 years ago (see Section 1.3.4). The role that is mentioned nowadays is failure detection, e.g., the notification of the crash of a process. However, failure notification does not need to be consistent to be able to solve agreement problems, e.g., consensus or atomic broadcast, as known from the failure detector approach [1.11]. Using group membership for solving atomic broadcast is an overkill. Moreover, since atomic broadcast does not require the consistent failure notification provided by group membership, one can wonder whether group membership should not be solved on top of atomic broadcast, as in [1.35], rather than the opposite (as done currently in many implementations). Group membership also allows the system to discard messages from output buffers. However, having one mechanism for solving two problems (failure notification and discarding messages from output buffers) is not a good solution [1.15].

Finally, the last issue that needs to be better addressed is the specification of *dynamic* group communication, i.e., group communication in an environment where processes can be added and removed during the computation[4]. One typical example is the *view synchronous multicast* primitive in the context of primary partition membership. The primitive appears close to *reliable broadcast*. However, both specifications are far away. The same holds for (static) *vs.* (dynamic) atomic broadcast. This is not satisfactory.

1.5 Conclusion

Results of more than twenty years of research have contributed to a very good understanding of many issues related to group communication. The need to ensure safety *and* liveness in the context of agreement problems (e.g., consensus, atomic broadcast) and the practical importance of consensus are now recognized. Nevertheless, group communication is not yet a solved problem. More work is needed on some issues, e.g., to come with convincing specifications of the membership problem (primary partition and partitionable), and with better specifications for dynamic group communication. The algorithmic benefit of the membership abstraction needs also to be better understood. Strong contributions are needed, to build on more solid grounds. Only so will group communication be better accepted outside of our community.

References

1.1 A. El Abbadi and S. Toueg. Availability in Partitioned Replicated Databases. In *ACM SIGACT-SIGMOD symp. on prin. of Database Systems*, pages 240–251, March 1986. See also *ACM Trans. on Database Systems*, 14(2):264-290, June 1989.

1.2 M. Aguilera, W. Chen, and S. Toueg. Failure detection and consensus in the crash-recovery model. In *Proceedings of the 12th International Symposium on Distributed Computing*, LNCS, pages 231–245. Springer, September 1998.

[4] [1.27] addresses the specification of *static* group communication primitives.

1.3 Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership Algorithms for Multicast Communication Groups. In *6th Intl. Workshop on Distributed Algorithms proceedings (WDAG-6), (LCNS, 647)*, pages 292–312, November 1992.

1.4 E. Anceaume, B. Charron-Bost, P. Minet, and S. Toueg. On the formal specification of group membership services. Technical Report 95-1534, Department of Computer Science, Cornell University, August 1995.

1.5 M. Barborak, M. Malek, and A. Dahbura. The Consensus Problem in Fault-Tolerant Computing. *ACM Computing Surveys*, 25(2):171–220, June 1993.

1.6 M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *proc. 2nd annual ACM Symposium on Principles of Distributed Computing*, pages 27–30, 1983.

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

1.8 K. Birman and T. Joseph. Exploiting Virtual Synchrony in Distributed Systems. In *11th Ann. Symp. Operating Systems Principles*, pages 123–138. ACM, Nov 87.

1.9 K. Birman, T. Joseph, T. Räuchle, and A. El Abbadi. Implementing Fault-Tolerant Distributed Objects. *IEEE Trans. on Software Engineering*, 11(6):502–508, June 1985.

1.10 T. D. Chandra, V. Hadzilacos, and S. Toueg. The Weakest Failure Detector for Solving Consensus. *Journal of ACM*, 43(4):685–722, 1996.

1.11 T. D. Chandra and S. Toueg. Unreliable Failure Detectors for Asynchronous Systems. In *proc. 10th annual ACM Symposium on Principles of Distributed Computing*, pages 325–340, 1991. See also *Journal of the ACM*, 43(2):225-267, 1996.

1.12 Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. On the impossibility of group membership. In *Proc. of the 15th ACM Symposium on Principles of Distributed Computing*, pages 322–330, Philadelphia, Pennsylvania, USA, May 1996.

1.13 J-M. Chang. Simplifying Distributed Database Systems Design by Using a Broadcast Network. In *Proc. of the ACM SIGMOD Int. Conference on Management of Data*, pages 223–233, June 1984.

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

1.15 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 *21st IEEE Symp. on Reliable Distributed Systems (SRDS-21)*, pages 244–249, Osaka, Japan, October 2002.

1.16 D. R. Cheriton and D. Skeen. Understanding the Limitations of Causally and Totally Ordered Communications. In *14th ACM Symp. Operating Systems Principles*, pages 44–57, Dec 1993.

1.17 D. R. Cheriton and W. Zwaenepoel. Distributed Process Groups in the V Kernel. *ACM Trans. on Computer Systems*, 2(3):77–107, May 1985.

1.18 G.V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Computing Surveys*, 4(33):1–43, December 2001.

1.19 F. Cristian, H. Aghili, R. Strong, and D. Dolev. Atomic broadcast: From simple message diffusion to byzantine agreement. In *IEEE 15th Int Symp on Fault-Tolerant Computing (FTCS-15)*, pages 200–206, June 1985.

1.20 Flaviu Cristian. Reaching Agreement on Processor Group Membership in Synchronous Distributed Systems. *Distributed Computing*, 4(4):175–187, April 1991.

1.21 D. Davcec and A. Burkhard. Consistency and Recovery Control for Replicated Files. In *Proceedings of the 10th Symposium on Operating Systems Principles*, pages 87–96, 1985.

1.22 X. Défago, A. Schiper, and P. Urban. Totally Ordered Broadcast and Multicast Algorithms: A Comprehensive Survey. TR DSC/2000/036, EPFL, Communication Systems Departement, October 2000.

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

1.24 A. Fekete, N. Lynch, and A.A. Shvartsman. Specifying and Using a Group Communication Service. *ACM Trans. on Computer Systems*, 19(2):171–216, May 2001.
1.25 M. Fischer. The Consensus Problem in Unreliable Distributed Systems (A Brief Survey). In *Int. Conf. on Foundations of Computation Theory (FCT)*, pages 127–140, 1983.
1.26 M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32:374–382, April 1985.
1.27 V. Hadzilacos and S. Toueg. Fault-Tolerant Broadcasts and Related Problems. Technical Report 94-1425, Department of Computer Science, Cornell University, May 1994.
1.28 B. Kemme. *Database Replication for Clusters of Workstations*. PhD thesis, ETH Zurich, Switzerland, 2000. Number 13864.
1.29 L. Lamport. Web home page. http://lamport.org/.
1.30 L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Comm. ACM*, 21(7):558–565, July 1978.
1.31 L. Lamport. Using Time Instead of Timeout for Fault-Tolerant Distributed Systems. *ACM Trans. on Progr. Languages and Syst.*, 6(2):254–280, April 1984.
1.32 L. Lamport. The Part-Time Parliament. Technical Report 49, Digital SRC, September 1989. See also *ACM Trans. on Computer Systems*, 16(2):133-169, May 1998.
1.33 L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Trans. on Progr. Languages and Syst.*, 4(3):382–401, July 1982.
1.34 D. Malkhi and M. Reiter. Byzantine Quorum Systems. *Distributed Computing*, 11(4):203–213, 1998.
1.35 P. M. Melliar-Smith, L. E. Moser, and V. Agrawala. Membership Algorithms for Asynchronous Distributed Systems. In *IEEE 11th Intl. Conf. Distributed Computing Systems*, pages 480–488, May 91.
1.36 M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of ACM*, 27(2):228–234, 1980.
1.37 F. Pedone. *The Database State Machine and Group Communication Issues*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, December 1999. Number 2090.
1.38 F. Pedone and A. Schiper. Generic Broadcast. In *13th. Intl. Symposium on Distributed Computing (DISC'99)*, pages 94–108. Springer Verlag, LNCS 1693, September 1999.
1.39 M. Rabin. Randomized Byzantine Generals. In *Proc. 24th Annual ACM Symposium on Foundations of Computer Science*, pages 403–409, 1983.
1.40 R. Reischuk. A New Solution for the Byzantine general's problem. Technical Report RJ 3673, IBM Research Laboratory, November 1982.
1.41 A. M. Ricciardi and K. P. Birman. Using Process Groups to Implement Failure Detection in Asynchronous Environments. In *Proc. of the 10th ACM Symposium on Principles of Distributed Computing*, pages 341–352, August 1991.
1.42 F.B. Schneider. Synchronization in Distributed Programs. *ACM Trans. on Progr. Languages and Syst.*, 4(2):125–148, April 1982.
1.43 R.D. Schlichting and F.B. Schneider. Fail-stop processors: an approach to designing fault-tolerant computing systems. *ACM Trans. on Computer Systems*, 1(3):222–238, August 1983.
1.44 J.H. Wensley. SIFT - Software Implementated Fault Tolerance. *FJCC*, pages 243–253, 1972.
1.45 J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shotak, and C.B. Weinstock. SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control. *Proc. IEEE*, 66(10):1240–1255, 1978.
1.46 M. Wiesmann. *Group Communications and Database Replication: Techniques, Issues and Performance*. PhD thesis, École Polytechnique Fédérale de Lausanne, Switzerland, 2002. Number 2577.