From Byzantine Agreement to Practical Survivability
A Position Paper

Dahlia Malkhi
School of Computer Science and Engineering
The Hebrew University of Jerusalem, Israel
dalia@cs.huji.ac.il

1. Introduction

Only a decade ago, issues of replication, high availability and load balancing were the focus of small, closely coupled cluster projects. Consequently, techniques for cluster management and small replication systems are abundant. However, the advent of the Internet led to wide spread and highly decentralized access of services and content that bring issues of scale and ubiquitous deployment. In particular, the need to maintain copies of replicated data consistent grows beyond the limits of any local cluster. Consequently, researchers have been looking at ways to improve scalability, survivability and dynamism of replication technology. Additionally, there are a number of recent application domains that exhibit new and challenging models for information replication. For example, advances in storage technology permit processes to share information by directly accessing data on disks that are connected to a *storage area network* (SAN), thereby avoiding going through a file system service (e.g., [22]). This form of direct data sharing necessitates coordination among processes contending for access to data, and presents new building blocks for doing it. New needs are also re-shaped by novel services such as Jini, a global resource discovery and location tool that allows anonymous and transient clients to be serviced; by Java-spaces, a universal shared data space; by Oceanstore [28], an eternal storage archive that is built of peers that have an economical incentive to cooperate; by Publius [50], an anonymous and survivable publishing archive; and others. Many other peer-to-peer (P2P) systems offer the potential of a truly survivable settings, but on the other hand, pose challenges of scale, dynamism and trust issues.

2. Principles of Replication

Replication is a fundamental component of computing systems that may occur even without explicit intention on the user's part. For example, when accessing a file over the network using the Network-File-System (NFS), a copy of the file's pages is created and kept on the local workstation. Likewise, when accessing the world-wide-web, copies of pages are brought from various replication and caching engines, e.g., content-delivery vehicles (such as `www.akamai.com`), proxy servers and from the browser's cache. In all of these examples, cache-copies are created automatically and transparently. In other settings, replication is done intentionally as a means for tolerating faults and for providing *high availability* of the data services despite failures. Whether explicit or not, the common denominator in all of these settings is the need for mechanisms that keep replicas consistent.

More specifically, consider the following problem: a data object x with methods $m_1(), .., m_k()$ operating on it. This is a general formulation, and x may contain both data and meta-data, may hold multiple data items, and so on. Several replica processes $r_1, ..., r_n$ store x, with the intention that the conglomerate of x copies should provide some benefits over a single copy. For example, it may provide high availability, i.e., when a single copy holding x fails it should be possible to continue invoking methods on x. It can provide fault tolerance, namely, recover from the failure of any single copy. Finally, it could provide load balance, so that each client interacts with a different copy of x. Despite this duplicity, methods invoked on x should behave as if a single copy exists–a notion known formally as *linearizabilty* [24]. Naturally, providing this behavior may introduce an overhead of replica maintenance, add complexity of consistency mechanisms, and increase the likelihood that some component in the system fails. There is a long standing effort to make replication costs acceptable in realistic settings, and some signs that success is already achieved in some domains. Many of these need to be re-thought today with the emergence of novel application areas.

The main paradigm for supporting replicated data is *active replication*, in which replicas execute the same sequence of methods on the object in order to remain consistent. A prerequisite for active replication to succeed is deterministic response, namely, that the sequence of methods invoked on an object determines the object's state. In particular, this means that there is no interference of scheduling and other environment parameters in determining the final outcome. This paradigm led to the definition of *State Machine Replication* (SMR), introduced by Lamport in a seminal paper [29] and overviewed later by Schneider [48]. A necessary building block of SMR is an engine that delivers method invocations reliably and in the same order to the replicas. This engine needs to form *agreement* on operation ordering, and is equivalent to the quintessential Byzantine agreement problem [32]. Replication using the SMR approach and agreement on operation ordering has undergone various enhancements and developments. In the remaining of the presentation, we review some of those that the author has been part of, as well as some others that help complement the picture.

2.1. Group Communication

One of the interesting approaches to achieve SMR is the *group communication* (GC) paradigm (see a collection of papers in [44] and a recent survey in [11]). In GC, replicated processes form agreement on the current configuration using *views*, and shift from configuration to configuration in response to changes (failures, joins). This allows to abstract away failures during normal operation and *optimistically* assume that no failures occur within a configuration.

The main advantage of the GC approach is that during stability periods, work within a configuration is highly efficient. From a theoretical point of view, optimism is what allows group communication systems to work very efficiently and escape the known two-round lower bound on agreement [46, 47]. (For a survey of lower bounds on agreement, see [26]). For example, a trivial protocol for SMR that works in **one round** within a stable configuration is as follows (borrowing from [29]). Each participant $r_1, ..., r_n$ sends a "proposed" operation (or null) in each round to all other participants, and after receiving proposals for the round from all other participants, orders the operations according to some pre-determined order (e.g., lexicographically). Similar, more commonly used variations of one-round protocols are leader-based or token-based protocols. In these variants, each round has a designated leader that may broadcast an operation to all and have it accepted immediately. In all of these one-round protocols, a process that partitions away from the system may deliver inconsistent operations with respect to the rest of the system, and would need to reconcile the conflict when it merges back with it. This weaker guarantee is the price of optimism in this approach.

A breakthrough in the semantics of GC systems was offered in ISIS in the Virtual Synchrony (VS) model [4]. The model defined a relationship between configuration maintenance and agreement on operation ordering. In a nutshell, VS requires that replicated processes that together move from one common configuration to a consecutive one, agree on operations ordered within the first configuration. Optimism is still manifested in the VS model with respect to processes that detach from the view, and hence the model permits efficient, one-round solutions. However, with careful attention to partitionable views semantics [3, 18, 25] and state transfer [2], reconciling split views is made easier using this model. Many systems today enjoy the benefits of GC support, including the Swiss Stock Exchange, the French Air and Traffic Control, The Fault-Tolerant CORBA (FT-CORBA) standard, the IBM AS-400 Cluster System, the IBM Phoenix project, the Microsoft NT Cluster, and others.

Though GC is efficient during stability periods, when failures occur and are detected the system needs to reconfigure. View maintenance requires solving agreement (on configuration changes), albeit amortizes the cost of agreement over potentially many operations ordering. Nevertheless, the drawback is that configuration maintenance scales poorly, as participants need to constantly monitor each other, yielding an n^2 probe complexity. Additionally, due to the high cost of configuration change it is not suitable for highly dynamic environments.

2.2. Quorum based replication

For systems that require stronger semantics the common approach is to guarantee safety of agreement at **all** times, while providing progress only during periods of system stability. The Paxos protocol [30] and its many variants (e.g., [31, 16, 33]) use this approach to achieve non-blocking fault tolerant agreement. The approach derives its fault tolerance capabilities from two mechanisms. The first one is a leader election oracle that during system stability periods guarantees the emergence of a unique leader. This captures the assumption that eventually, communication is timely and failures cease for sufficiently long to allow progress. The second means is a *quorum system* for a built-in fault tolerance of an a priori, fixed threshold on the number of possible failures. A quorum system is a set of subsets of the replicas, such that each operation can be performed only on a subset (a *quorum*). Quorum systems are means for enhancing the fault tolerance and load balance of replicated services. (See, e.g., [37] for an overview.)

In order to introduce the Paxos approach to agreement and operation ordering, we use a recent simple formulation of [5, 12]. Processes together emulate an abstraction of a *ranked register*, on which two types of wait-free operations are allowed:

1. A write operation $W = \text{rr-}write(r, value)$ tries to write $value$ with rank r. The operation may *abort* if

some higher ranked $R =$ rr-$read$ has been invoked before the completion of W, and otherwise, it *commits*.

2. A read operation $R =$ rr-$read(r)$ reads a pair $\langle v', r' \rangle$ that was previously rr-$write$'en, such that if any lower-ranked rr-$write(r'', v'')$ (ever) commits, $r'' < r$, it is "seen by" R, i.e., $r' \geq r''$.

Reaching agreement with a ranked-register abstraction is almost immediate: A participant r_i invokes any eventually-exclusive leader protocol, to guarantee progress. The leader chooses a unique rank r, and then invokes rr-$read$ followed by a rr-$write$. The write operation attempts to commit the value that was read, if any, or r_i's own input.

Implementing rr-$read()$ and rr-$write()$ is also quite straight-forward, and entails each a two-round message exchange. Each participant stores a value and a rank. The rr-$read$ operation obtains the highest ranked value from a quorum of participants. The rr-$write$ operation contacts a quorum with a new value and rank. When contacted with a rr-$write(r, v)$ request from another process, a participant rejects the request if r is lower than its stored rank, and otherwise updates both the recorded value and the rank of the ranked register. The rr-$write$ commits if it is accepted by a quorum.

A commonly used variant of Paxos, the revolving-token protocol (e.g., see [9, 35]), has a similar principle but different formulation. In this form, the leader election is explicitly coded into the algorithm and its liveness is expressed using failure detection conditions. The agreement protocol advances in virtual rounds. The leader is implicitly determined by a round number. Instead of contending for leadership, processes move from a round to the next when they *suspect* the current round's leader. The eventual unique-leader condition is guaranteed when a round's leader is alive and not suspected by anyone.

2.3. Survivability methods

In addition to tolerating benign faults, a shift toward systems that tolerate arbitrary *corruption* of participants has become necessary for critical applications and for a wide spread deployment. Both the GC paradigm and the quorums paradigm underwent this shift. The Rampart [45] and SecureRing [27] systems are examples of secure GCs. In this setting, one round ordering is not possible since a corrupt leader (token holder) might disperse inconsistent messages. Bracha and Toueg [6] solve this using a reliable broadcast protocol that guarantees agreement on the contents of each message sent by the leader. The principles of their broadcast primitive underlie most existing solutions, and are as follows: Let t be a presumed threshold on the number of faulty participants in $\{r_1, ..., r_n\}$, where $t < n/3$. In order to deliver a message m reliably and uniformly by $\{r_1, ..., r_n\}$, each r_i that receives m echoes the received message; upon receiving echoes of m from $2t + 1$ processes, m is proposed for commitment; and upon receiving commitments from $t + 1$, it can be delivered. Signatures replace in Rampart [45] one round of messages, and round costs are amortized in SecureRing [27].

A recognized problem with the GC approach for survivability is that since the system may reconfigure over and over, a corrupt collusion might gradually cause the removal of too many correct participants and assume control over the system. Therefore, in Byzantine settings, there is an inherent advantage to the quorum based approach is that the resilience threshold is a priori fixed.

In the context of the quorums paradigm, Byzantine quorum systems were introduced in [38], extending quorum replication techniques to cope with corrupt participants. Intuitively, Byzantine quorum systems require operations to intersect in sufficiently many **correct** processes to mask out the behavior of faulty ones. More formally, a t-masking quorum system designed to tolerate t failures is required to satisfy two properties:

Availability For any possible set of t faults, there exists a quorum that does not contain any failed process.

Safety Each pair of quorums has intersection of size at least $2t + 1$.

Byzantine quorums can be surprisingly efficient, requiring operations to access only $O(\sqrt{n}\sqrt{t})$ of the system [41].

Replicating data with Byzantine quorum systems requires that values stored in x are obtained from at least $t + 1$ copies of x. Consequently, replication is significantly more involved than in the benign failure case, since reading x might snapshot partially completed updates and is not guaranteed to obtain $t + 1$ identical copies. The first adaptation of Paxos-like replication to the Byzantine setting was provided in [7], using the revolving token paradigm. A quorum-based implementations of operation ordering is given in [13] and an atomic object emulation that makes use of Byzantine quorums for collusion-resistance is found in [13, 42].

2.4. From Process Replication to Data-Centric Paradigms

The SMR approach is a process-centric approach, in which processes actively participate in active replication protocols. In contrast, in a data-centric paradigm replicas may simply store data and operate methods on it. This data centric view is attractive in two independent settings. One is the setting of *storage area networks* (SAN). The SAN technology enables cost-effective bandwidth scaling by allowing data to be transferred directly from network attached disks to clients so that the file server bottleneck is eliminated. Since clients (or a group of designated SAN servers) need to coordinate and secure their accesses to disks, they need to implement distributed access control and locking for the disks. This leads to the usage of SMR as is done, e.g., in Compaq's Petal [34] and Frangipani [49], using disks as memory for information sharing and coordination. The memory objects provided in a SAN may include high level objects that take advantage of extended functionality of *Active Disks* (see, e.g., [22]). In particular, specialized functions that require specific semantics not normally provided by drives can be provided by remote functions on Active Disks. Examples include a *read-modify-write* operation, or an atomic *create* that both creates a new file object and updates the corresponding directory object. Such advanced operations are already used for optimization of higher-level file systems such as NFS on NASD [23]. Adaptations of the Paxos protocol to the disk model are given in [21] and a scalable variation that takes advantage of Active Disks is found in [12].

The data-centric approach faithfully represents another realistic setting, the classic client-server model, with a potentially very large and dynamic set of clients. This is the setting for which scalable systems like the Fleet object repository [39, 40] were designed. In this setting, a highly available service is implemented by a replicated set of servers, a threshold of which may be faulty. Replicas that store objects may be numerous and highly decentralized. Hence, the system should be designed to avoid server-to-server interaction and monitoring. Additionally, for the sake of survivability, server logic needs to be kept simple and well understood. This leads to the same data-centric design as above, in which servers simply store data and execute methods on it. This paradigm provides for coordination and information sharing among transient clients through the group of servers, while not requiring servers to interact among themselves, and it avoids the complexity of failure monitoring and reconfiguration. A replication protocol for this setting was provided in [13].

The design also lends itself to other important emerging areas. One of the promising applications of this approach is efficient unbridged access to dedicated database servers (such as SQL servers). The functionality of such systems is typically limited to that of database query processing, because it is considered infeasible to run application protocols on such systems. The way in which the industry currently deals with this issue is by adding another middle-tier application server layer whose responsibility is to handle high-level application protocols. The data centric approach allows to eliminate the middle-tier altogether, in order to save on communication and prevent the middle-tier server from becoming a bottleneck. The advantage in using the fixed databases to share information is that the client processes will not need to communicate with one another, nor to monitor changes and respond in any way to a reconfiguration.

Another dimension in which these new services differ from traditional clustering applications is in the aim to serve a potentially huge set of clients, who may be transient, mobile, and are unknown in advance. It is inconceivable that such clients will all use the same level of service or the same set of servers. Hence, different objects should carve for themselves distinct replication universes, tuned to various needs in terms of size, cost, and quality of service. For example, one client may store a critical piece of information with high redundancy while another client, storing a very large file, utilizes only low degree of redundancy. Dynamic replication services are also instrumental in seamlessly transforming legacy objects into replicated servers in systems like quorum-based FT-CORBA [10] and Rambo [36].

Data centric replication is readily adaptable to the Byzantine setting using Byzantine quorum systems [38], e.g., utilizing the replication protocols of [12, 42]. An interesting question is whether in this model, Byzantine *clients* might cause inconsistency, and whether servers should therefore corroborate client operations between themselves to enforce consistency. The approach initiated the Fleet system [39, 40] and manifested in several works that follow it [13, 10, 42] indicates that this need not be the case. The reasons for this is that standard access control mechanisms prevent unauthorized clients from accessing non-faulty replication servers. If, nevertheless, malicious clients obtain authorization to access data, they may cause arbitrary updates to data in any case, and hence, it makes little sense to enforce consistency on such clients' operations.

2.5. Failure Detection Mechanisms

The last development we mention is in the realization of failure detection mechanisms. For many years, researchers have struggled with the uncertainty of failure detection and the consequential impossibility of reaching agreement in the face of failures and asynchrony [20]. The conditions for guaranteeing progress despite failures are well known (see [17] for a survey of different models), and essentially require that a single coordinator be eventually accepted by all. These conditions have been formalized in a variety of ways, including assumptions on partial synchrony [15] and abstract failure detection conditions [9, 8]. Additionally, a well known approach for circumventing impossibilities is via randomization (see a survey of randomization techniques for consensus in [14]).

A number of recent efforts address the challenge of actually building the tools that facilitate progress in coordination protocols. The *fail-awareness* mechanisms of Fetzer et al. [19] make use of hardware timeout mechanisms and remote shut-down facilities to construct fail-awareness protocols. *Heart-bit failure detectors* [1] require that out of an infinite stream of messages, infinitely many make it to their target, and make use of this property to facilitate failure detection. Randomization was recently harnesses directly in [13] in an eventually safe leader election primitive, using randomized backoff techniques borrowing from the well known multiple-access networks domain. And finally, an *ordering oracle* was defined in [43] that makes use of network broadcast (or multicast) and achieves coordination without any time considerations.

References

[1] M. Aguilera, W. Chen and S. Toueg. Using the Heartbeat failure detector for quiescent reliable communication and consensus in partitionable networks. *Theoretical Computer Science* 220(1):3–30, June 1999.

[2] Y. Amir, G. V. Chockler, D. Dolev and R. Vitenberg. Efficient state transfer in partitionable environments. In Proceedings of *the European Research Seminar on Advances in Distributed Systems* (ERSADS97), pages 183-191, Switzerland, March 1997.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership algorithms for multicast communication groups. In Proceedings of the *6th Intl. Workshop on Distributed Algorithms* (WDAG-6), (LNCS, 647) Haifa, Israel, Nov. 1992, pp. 292-312.

[4] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In Proceedings of the *11th Annual Symposium on Operating Systems Principles*, pages 123–138, November 1987.

[5] R. Boichat, P. Dutta, S. Frolund and R. Guerraoui. Deconstructing Paxos. *Technical Report DSC ID:200106*, Communication Systems Department (DSC), École Polytechnic Fédérale de Lausanne (EPFL), January 2001. Available at `http://dscwww.epfl.ch/EN/publications/documents/tr01_006.pdf`.

[6] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM* 32(4):824–840, October 1985.

[7] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation - OSDI'99*, February, 1999, New Orleans, LA.

[8] T. D. Chandra, V. Hadzilacos and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM* 43(4):685–722, July 1996.

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

[10] G. Chockler, D. Dolev and D. Malkhi. A quorum based approach to CORBA fault-tolerance. In Proceedings of the *European Research Seminar on Advances in Distributed Systems* (Ersads 2001), Bologna, Italy, May 2001.

[11] G. V. Chockler, I. Keidar and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computing Surveys* 33(4):1–43, December 2001.

[12] G. Chockler, D. Malkhi. Active disk Paxos with infinitely many processes. In Proceedings of the *21st ACM Symposium on Principles of Distributed Computing* (PODC '02), August 2002. To appear.

[13] G. Chockler, D. Malkhi, and M. K. Reiter. Backoff protocols for distributed mutual exclusion and ordering. In Proceedings of the *21st International Conference on Distributed Computing Systems*, pages 11-20, April 2001.

[14] B. Chor and C. Dwork. Randomization in Byzantine agreement. *Advances in Computing Research, Randomness in Computation*, volume 5, JAI Press, edited by S. Micali, pp. 443–497, 1989.

[15] F. Cristian and C. Fetzer. The Timed Asynchronous distributed system model. In Proceedings of the *28th Annual International Symposium on Fault-Tolerant Computing*, June 1998.

[16] R. DePrisco, B. Lampson and N. Lynch. Fundamental study: Revisiting the Paxos algorithm. *Theoretical Computer Science* 243:35–91, 2000.

[17] D. Dolev, C. Dwork and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM* 34(1):77–97, January 1987.

[18] D. Dolev, D. Malki and R. Strong, Framework for partitionable operation. **Brief announcement** in Proceeding of the *15th ACM Symposium on the Principles of Distributed Computing* (PODC 96), Philadelphia, 1996, page 343. Full version available as TR-CS 95-4, Hebrew University of Jerusalem, February 1995.

[19] C. Fetzer and F. Cristian. Fail-awareness: An approach to construct fail-safe applications. Journal of Real-Time Systems, to appear.

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

[21] E. Gafni and L. Lamport. Disk Paxos. In Proceedings of *14th International Symposium on Distributed Computing (DISC'2000)*, pages 330–344, October 2000.

[22] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg and J. Zelenka. A cost-effective high-bandwidth storage architecture. In Proceedings of the *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 1998.

[23] G. A. Gibson, D. F. Nagle, K. Amiri, F. W. Chang, H. Gobioff, E. Riedel, D. Rochberg and J. Zelenka. Filesystems for network-attached secure disks. *Technical Report CMU-CS-97-118*, July 1997.

[24] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems* 12(3):463–492, July 1990.

[25] I. Keidar and D. Dolev. Totally ordered broadcast in the face of network partitions: exploiting group communication for replication in partitionable networks. In *Dependable Network Computing, Chapter 3*, D. Avresky Editor, Kluwer Academic Publications. January 2000.

[26] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults - a tutorial. MIT Technical Report MIT-LCS-TR-821, May 24, 2001. (Preliminary version in SIGACT News 32(2), Distributed Computing column, pages 45-63, June 2001.

[27] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing protocols for securing group communication. In *Proceedings of the 31st Annual Hawaii International Conference on System Sciences*, 1998.

[28] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In Proceedings of the *Ninth international Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS 2000), November 2000.

[29] L. Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of the ACM* 21(7):558–565, July 1978.

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

[31] L. Lamport. Paxos made simple. *Distributed Computing Column of SIGACT News* 32(4):34–58, December 2001.

[32] L. Lamport, R. Shostak and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems* 4(3):382–401, July 1982.

[33] B. W. Lampson. How to build a highly available system using Consensus. In Proceedings of the *10th International Workshop on Distributed Algorithms (WDAG)*, Springer-Verlag LNCS 1151:1-17, Berlin, 1996.

[34] E. K. Lee and C. Thekkath. Petal: Distributed virtual disks. In Proceedings of the *7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 84-92, October 1996.

[35] W. K. Lo and V. Hadzilacos. Using failure detectors to solve consensus in asynchronous shared-memory systems. In Proceedings of the *8th International Workshop on Distributed Algorithms (WDAG)*, Springer-Verlag LNCS 857:280-295, Berlin, 1994.

[36] N. Lynch and A. Shvartsman. RAMBO: A reconfigurable atomic memory service for dynamic networks. In Proceedings of *the 16th International Symposium on DIStributed Computing* (DISC) October 2002, Toulouse, France. To Appear.

[37] D. Malkhi. Quorum systems. Chapter in *The Encyclopedia of Distributed Computing*, Joseph Urban and Partha Dasgupta, editors, Kluwer Academic Publishers. To be published. `http://www.cs.huji.ac.il/~dalia/pubs/quorums.ps.gz`.

[38] D. Malkhi and M. Reiter, Byzantine quorum systems. *The Journal of Distributed Computing* 11(4):203–213, 1998.

[39] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering* 12(2):187–202, March/April 2000.

[40] D. Malkhi, M. K. Reiter, D. Tulone and E. Ziskind. Persistent objects in the Fleet system. In Proceedings of *DARPA's second DARPA Information Survivability Conference and Exposition* (DISCEX II), 2001.

[41] D. Malkhi, M. Reiter and A. Wool. The load and availability of Byzantine quorum systems. *SIAM Journal of Computing* 29(6):1889–1906, 2000.

[42] J.P. Martin, L. Alvisi, M. Dahlin. Minimal byzantine storage. In Proceedings of *the 16th International Symposium on DIStributed Computing* (DISC) October 2002, Toulouse, France. To Appear.

[43] F. Pedone, A. Schiper, P. Urban and D. Cavin. Solving agreement problems with weak ordering oracles. Technical Report IC/2002/010, Ecole Polytechnique Federale de Lausanne, Switzerland, March 2002.

[44] D. Powell, editor. Group communication. *Communications of the ACM* 39(4), April 1996.

[45] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, November 1994.

[46] M. D. Skeen. Nonblocking commit protocols. In *SIGMOD International Conference Management of Data*, 1981.

[47] M. D. Skeen. Crash recovery in a distributed database system. PhD thesis, UC Berkeley, May 1982.

[48] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 22(4):299–319, December 1990.

[49] C. Thekkath, T. Mann and E. K. Lee. Frangipani: A scalable distributed file system. In Proceedings of the *16th ACM Symposium on Operating Systems Principles*, pages 224–237, October 1997.

[50] M. Waldman, A. Rubin and L. Crabor. Publius, A robust, tamper-evident and censorship-resistant web publishing system. In Proceedings of the *9th USENIX Security Symposium*, August, 2000.