Refined Quorum Systems

[Position Paper] *

Rachid Guerraoui
IC, EPFL
CH-1015, Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Marko Vukolic
IC, EPFL
CH-1015, Lausanne, Switzerland
marko.vukolic@epfl.ch

ABSTRACT

It is considered good distributed computing practice to devise object implementations that tolerate contention, periods of asynchrony and a large number of failures, but perform fast if few failures occur, the system is synchronous and there is no contention. This paper initiates the first study of quorum systems that help design such implementations. Namely, our study of quorum systems encompasses, at the same time, the optimal resilience of distributed object implementations (just like traditional quorum systems), as well as their *optimal best-case complexity* (unlike traditional quorum systems).

We introduce the notion of a *refined quorum system* (RQS) of some set S as a set of three refined classes of subsets (quorums) of S: first class quorums are also second class quorums, which are also third class quorums. First class quorums have large intersections with all other quorums, second class quorums might have slightly smaller intersections with those of the third class, the latter simply correspond to traditional quorums. Intuitively, under uncontended and synchronous conditions, a distributed object implementation would expedite an operation if a quorum of the first class is accessed, then degrade gracefully depending on whether a quorum of the second or the third class is accessed. Moreover, we show that our RQS is, in a sense, minimal (i.e., necessary and sufficient), for optimally resilient and best-case optimal implementations of two fundamental Byzantine-resilient objects — *atomic storage* and *consensus*.

RQS are devised assuming a general adversary structure, and this basically allows algorithms relying on RQS to relax the assumption of independent process failures, often questioned in practice.

1. INTRODUCTION

Quorum systems are powerful mathematical tools to reason about distributed implementations of shared objects including read/write storage (e.g., [3, 16, 22]) and consensus [8, 18, 28]. In particular, quorum systems have been used to reason about implementations that tolerate asynchrony and are optimally resilient to process failures. Originally, a quorum system was defined as a set of subsets that intersect [9], and this notion was key to reasoning about crash-resilient asynchronous algorithms. More sophisticated forms of quorum systems have been introduced to cope with Byzantine (malicious) failures [21]: these require larger intersections among subsets (i.e., quorums) [22].

Perhaps surprisingly, most recent distributed object implementations, e.g., [1, 4, 5, 7, 10, 11, 20, 23, 26, 29] make little use of an abstract quorum notion. The absence of such a notion makes it in particular difficult to move away from a threshold-based adversary structure with the assumptions of independent and uniformly distributed failures, often questioned in practice, to a general adversary structure. The reason for this absence is, we believe, because traditional quorum notions (be they simple or Byzantine), while very useful to reason about the resilience dimension, are not adequate to capture the complexity dimension, specifically the *best-case* one. The implementations in [1,4,5,7,10,11,20,23, 26,29] were indeed devised to tolerate worst-case conditions, namely a large number of failures, arbitrarily long periods of asynchrony and contention. Motivated by practical considerations however, these implementations are also *optimistic* and geared to reduce best-case complexity, i.e., performance under situations of synchrony and no-contention, which are typically argued to be frequent in practice. As a consequence of their optimism, these implementations expedite operations in uncontended and synchronous situations, provided *"enough"* servers are accessed. Precisely capturing this very notion of *"enough"* in general terms was the motivation of this work.

This paper introduces the notion of *refined quorum systems* (RQS). In short, a refined quorum system of some set of elements S is a set of three classes of subsets (quorums) of S: first class quorums are also second class quorums, which are also third class quorums. Quorums (subsets of S) of the first class have large intersections with quorums of other classes, those of the second class might have slightly smaller intersections with those of the third class, the latter simply correspond to traditional quorums. In the context of a distributed object implementation, a set S would typically be the set of fault-prone server processes over which some object abstraction (e.g., storage or consensus) is implemented.

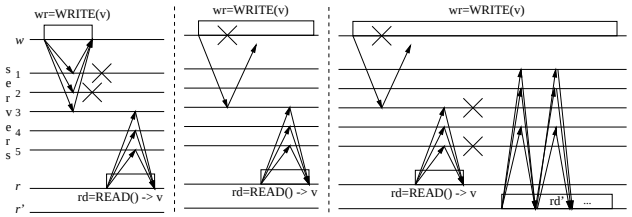
1.1 Example

To illustrate the intuition behind refined quorums, con-

*Full paper is available as a EPFL Technical Report LDP-REPORT-2007-002 (submitted for publication). We thank Hagit Attiya, Christian Cachin, Petr Kouznetsov and Eric Rupert for their very helpful comments.

sider the simple context of a crash-resilient implementation of an atomic storage over a set of server processes [3]. It is known [6] that no optimally resilient atomic storage algorithm can have both reads and writes complete in a single communication round-trip (we simply say round), even if a single writer is involved (SWMR). For instance, the classical, optimally crash-resilient solution [3] (that assumes a majority of correct processes) requires two rounds for a read.

As we discussed earlier, it is practically appealing to look into best-case complexity and ask if it is possible to expedite *both* reads and writes within a single round in a synchronous and contention-free period. Clearly, if the reader (resp. the writer) access all servers in the first round, then it can immediately return a valid response. But do we need to access all servers? How many servers actually *need* to be accessed to achieve such a *fast termination* in best-case conditions?



(a) ex2 (b) ex3 (c) ex4

Figure 1: Violation of atomicity in case the single-round operations access only 3 servers.

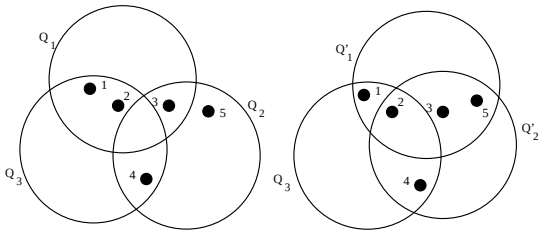
Consider the following simple example of 5 servers that implement a crash-tolerant atomic storage assuming $t = 2$ server failures (optimal resilience). We argue below that any algorithm that greedily expedites read/write operations in one round during synchronous and contention-free periods whenever $S - t = 3$ servers are accessed, violates atomicity. This is depicted through several executions of such an algorithm (Figure 1):

1. In $ex1$, the writer w invokes $wr = \mathsf{write}(v)$ and servers 4 and 5 are faulty. Then, wr writes the value v into the subset of servers $Q_1 = \{1, 2, 3\}$ and completes in a single round.

2. $Ex2$ (Fig. 1(a)) is slightly different because servers 4 and 5 are actually correct. Yet wr also completes in a single round, after writing in Q_1. Then servers 1 and 2 crash and a read rd (by the reader r) is invoked. Assuming synchrony and no contention, rd accesses server set $Q_2 = \{3, 4, 5\}$ and completes in a single round.

3. $Ex3$ (Fig. 1(b)) is similar to $ex2$ except that (1) the write is incomplete and writes only to server 3, (2) servers 1 and 2 (i.e., servers from the set $Q_2 \setminus Q_1$) are correct, but the communication between the reader and the servers from $Q_2 \setminus Q_1$ is delayed. Read rd does not distinguish $ex3$ from $ex2$ and completes in a single round, returning v.

4. Finally, consider $ex4$ (Fig. 1(c)) that extends $ex3$ by: (1) the crash of servers 3 and 5 and (2) the invocation of read rd' by a different reader r'. This reader cannot return v using $Q_3 = \{1, 2, 4\}$ regardless of how many rounds are used. Atomicity is violated.



(a) $|Q_1| = |Q_2| = 3$ (b) $|Q'_1| = |Q'_2| = 4$

Figure 2: Quorum intersections

Essentially, atomicity is violated because $Q1 \cap Q2 \cap Q3 = \emptyset$ (Figure 2(a)). On the other hand, we can devise a storage algorithm that achieves fast termination whenever 4 servers are accessed. For instance:

- A write wr completes in a single round only if it writes v to 4 servers, say $Q'_1 = \{1, 2, 3, 5\}$. A subsequent single-round read rd will also have to access at least 4 servers, say $Q'_2 = \{2, 3, 4, 5\}$ (including at least 3 servers from Q'_1). A subsequent read rd' that accesses some subset Q_3 of 3 servers will surely learn about v since there is a set $X = Q'_1 \cap Q'_2$ of (at least) 3 servers that witnessed both wr and rd, and X intersects with any set of 3 servers. This server in the intersection will inform rd' about the value written by wr.

The key to atomicity is that $Q'_1 \cap Q'_2 \cap Q_3 \neq \emptyset$. Namely, (Figure 2(b)) in a system of 5 elements, any two subsets of 4 elements intersect with any subset of 3 elements. Basically, boosting complexity requires to access subsets of servers that have larger intersections than traditional quorums. The above example is (relatively) simple because we were interested in the necessary and sufficient intersection properties considering: a) crash failures, b) threshold-based quorums and c) no graceful degradation.

The idea behind our notion of *refined quorum system* is precisely to characterize the required intersection properties in a precise and general manner. We aim at a characterization that is necessary and sufficient for optimizing the best-case complexity of various distributed object implementations, in various failure models, under various adversary structures, and also considering graceful degradation.

1.2 Contributions

Intuitively, under uncontended and synchronous conditions, a distributed object implementation would expedite an operation if a quorum of the first class is available, then degrade gracefully, depending on whether a quorum of the second or the third class is available. We argue that our quorum notion is, in a sense, complete: there is no reason for further refinement of quorums with the goal of optimizing best-case efficiency, since the properties provided by our third class quorums are anyway necessary for hindering the partitioning of the asynchronous system, which is key to any resilient distirbuted object implementation.

Our refined quorum systems are designed to handle a general adversary structure expressing situations where an adversary controls subsets of processes in a specific manner [15, 22]. As a consequence, this allows algorithms designed with such quorums to relax the assumption of independent process failures, often criticized in practice. In

the full paper [13] we illustrate the power of our notion of RQS by introducing two new atomic object implementations. Each algorithm is interesting in its own right and is, in a precise sense, the first fully optimal protocol of its kind.

- Our first object implementation is a new Byzantine-resilient asynchronous distributed storage algorithm. Such algorithms constitute an active area of research and are appealing alternatives to classical centralized storage systems based on specialized hardware [27]. The challenge when devising storage algorithms is to ensure that *reads* and *writes* have low latency in most frequent situations, while (a) tolerating the failures of a large number of base servers (typically commodity disks) as well as any number of clients that access the storage (wait-freedom [14]) and (b) ensuring strong consistency (ideally atomicity [17]). Using RQS, we present an atomic wait-free storage algorithm that combines optimal resilience with the lowest possible *read/write* latency in best-case conditions (synchrony and no-contention). Under such conditions, our algorithm expedites storage operations (*reads* and *writes*) in a single round if a first class quorum is accessed, in two rounds if a second class quorum is accessed and in three rounds otherwise. The latter case is when a third class quorum is available which is a necessary condition for resilience anyway. Our algorithm does not use any data authentication primitive, and matches the resilience and complexity lower bounds of [11, 24] when these are extended to a general adversary structure, together with a new complementary bound. Our new bound captures the best-case complexity of gracefully degrading atomic storage implementations.
- Our second algorithm implements a Byzantine-resilient consensus abstraction in the general state machine replication (SMR) framework of [18], distinguishing different process roles: *proposers* that propose values to be learned by *learners* with the mediation of *acceptors*. Our algorithm is the first to tolerate (1) any number of Byzantine failures of proposers and learners, (2) the largest possible number of acceptor failures, and (3) arbitrarily long periods of asynchrony. On the other hand, under best-case conditions, our algorithm allows a value to be learned in only two message-delays in case a first class quorum is accessed, and in three (resp., four) message delays in case a second (resp., third) class quorum is accessed. Note here that (a) learning in a single message delay is obviously impossible with multiple or potentially Byzantine proposers, and (b) the availability of a third class quorum is anyway necessary for resilience. Our algorithm matches the resilience and complexity lower bounds of [19] when these are extended to a general adversary structure, together with a new complementary bound on consensus algorithms that degrade gracefully in best-case executions. These bounds state minimal conditions under which the SMR approach can be made optimally resilient and best-case efficient. Until now, it was not clear whether the conditions of [19] were also sufficient. We show they are and we complement them.

In this position paper we first present our quorum notion and illustrate how it generalizes previous ones through examples from the literature. Then, we point out some open research directions. We postpone the detailed model as well as our algorithms and their proofs to the full paper [13].

2. REFINED QUORUM SYSTEMS

Definition of our refined quorum system is expressed in an environment including S a non-empty set of elements, and an *adversary structure* (or, simply, *adversary*) $\boldsymbol{B}$ defined as follows [15]. Let $\boldsymbol{B}$ be any set of subsets of S. $\boldsymbol{B}$ is an *adversary* (for the set S) if: $\forall B \in \boldsymbol{B}$: $B' \subseteq B \Rightarrow B' \in \boldsymbol{B}$.

Let $\boldsymbol{RQS}$ be any set of subsets of S.

Definition 1. **Refined Quorum System.** We say that $\boldsymbol{RQS}$ is a *refined quorum system* for a set S and adversary $\boldsymbol{B}$, if $\boldsymbol{RQS}$ has two subsets $\boldsymbol{QC_1} \subseteq \boldsymbol{QC_2} \subseteq \boldsymbol{RQS}$ such that the following properties hold: (every $\boldsymbol{QC_i}$ is called a *quorum class*, and elements of $\boldsymbol{QC_i}$ are called *class i elements*)

Property 1. An intersection of any two elements of $\boldsymbol{RQS}$ is not an element of $\boldsymbol{B}$, i.e.,

- $\forall Q, Q' \in \boldsymbol{RQS}$: $Q \cap Q' \notin \boldsymbol{B}$.

Property 2. The intersection of any two class 1 elements and any element of $\boldsymbol{RQS}$ is not a subset of the union of any two elements of $\boldsymbol{B}$, i.e.,

- $\forall Q_1, Q_1' \in \boldsymbol{QC_1}$, $\forall Q \in \boldsymbol{RQS}$, $\forall B_1, B_2 \in \boldsymbol{B}$: $Q_1 \cap Q_1' \cap Q \nsubseteq B_1 \cup B_2$.

Property 3. The intersection of any class 2 element Q_2 and any element Q of $\boldsymbol{RQS}$ is:
(a) not a subset of the union of any two elements of $\boldsymbol{B}$ (we say $P_{3a}(Q_2, Q)$ holds), *or*
(b) its intersection with every class 1 element[1] is not an element of $\boldsymbol{B}$ (we say $P_{3b}(Q_2, Q)$ holds), i.e.,

- $\forall Q_2 \in \boldsymbol{QC_2}$, $\forall Q \in \boldsymbol{RQS}$, $\forall B_1, B_2 \in \boldsymbol{B}$: $(Q_2 \cap Q \nsubseteq B_1 \cup B_2) \vee$ $\vee (\boldsymbol{QC_1} \neq \emptyset \wedge \forall Q_1 \in \boldsymbol{QC_1}$: $|Q_1 \cap Q_2 \cap Q| \notin \boldsymbol{B})$.

We simply call elements of a refined quorum system — *quorums*. In addition, for simplicity, we sometimes refer to any quorum that is not a class 2 quorums as a *class 3* quorum, and write $\boldsymbol{QC_3} = \boldsymbol{RQS}$. Note that class 1 quorums are also class 2 quorums, which are also class 3 quorums. Notice also that, when $\boldsymbol{QC_1} = \boldsymbol{QC_2}$, Property 2 implies Property 3. Furthermore, when $\boldsymbol{B} = \emptyset$, Property 1 implies Property 3. Therefore, Property 3 is interesting on its own only if $\boldsymbol{B} \neq \emptyset$ and $\boldsymbol{QC_1} \neq \boldsymbol{QC_2}$.

2.1 Examples

We denote by $\boldsymbol{B_k}$ a *k-bounded threshold adversary*, a special case of an adversary that contains all subsets of S with cardinality at most k (i.e., $\boldsymbol{B_k} = \{B | B \subseteq S \wedge |B| \leq k\}$). Moreover, we denote by $\boldsymbol{Q_i}$ the set of subsets of S that contains all subsets of S that contain all but at most i elements of S, i.e., $\boldsymbol{Q_i} = \{P | P \subseteq S \wedge |P| \geq |S| - i\}$.

Example 1. Figure 3 depicts a simple illustration of a RQS for an adversary $\boldsymbol{B_1}$: 4 quorums are involved. As depicted by the example, the cardinality of a quorum might not be a good indication of its class: it is the intersection with others that matters. Quorum Q_1 contains 5 elements and is a class 1 quorum, while Q' contains 6 elements yet is only an ordinary (or class 3) quorum.

[1]Assuming there is at least one class 1 element, i.e., $\boldsymbol{QC_1} \neq \emptyset$.

To see that the quorum system in Figure 3 is indeed a RQS, note that every pair of depicted sets intersects in at least 2 elements (satisfying Property 1). Q_1 intersects with every other set in at least 3 elements (satisfying Property 2, for an intersection with itself). Moreover, $P_{3a}(Q_2, Q')$ and $P_{3a}(Q_2, Q_1)$ hold (since $|Q2 \cap Q'| = 2 = |Q2 \cap Q1|$) as well as $P_{3b}(Q_2, Q)$ (since $|Q_2 \cap Q \cap Q_1| = 2$). Hence, $\boldsymbol{RQS} = \{Q, Q', Q_2, Q_1\}$ is a refined quorum system, where Q_1 (resp., Q_2) is a class 1 (resp., class 2) quorum.

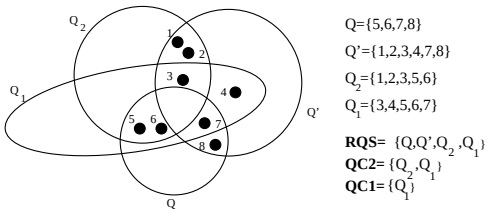


Figure 3: A simple example of a RQS

In the following, we give more illustrations of our quorum notion by explaining how it generalizes traditional ones. In the following, an adversary $\boldsymbol{B}$ for a set of processes S contains all subsets of S that can simultaneously be Byzantine. Note that, in this paper, we do not consider a process that simply crashes to be Byzantine.

Example 2. Consider the case where: (a) $\boldsymbol{B} = \emptyset$, (b) $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \emptyset$ and (c) $\boldsymbol{RQS} = \boldsymbol{Q}_{\lfloor(|S|-1)/2\rfloor}$ (i.e., every majority subset of S is a quorum). Property 1 is trivially satisfied. So are Properties 2 and 3, since $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \emptyset$. This quorum system is typically used when devising algorithms that tolerate (a minority of) crash-failures, e.g., [3, 9, 18, 25].

Example 3. Consider the case of an adversary $\boldsymbol{B}_{\lfloor|S|/3\rfloor}$, where (a) $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \emptyset$ and (b) $\boldsymbol{RQS} = \boldsymbol{Q}_{\lfloor(|S|-1)/3\rfloor}$. In this case, each quorum contains more than two thirds of processes and satisfies Property 1. Properties 2 and 3 are also satisfied (since $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \emptyset$). Such a quorum system is typically used to tolerate (up to one third of) Byzantine failures, e.g., [4, 5, 21, 24].

Example 4. The RQS for which $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \emptyset$ is a *disseminating quorum system* in the sense of [22]. In [22], disseminating quorum systems are used to build resilient distributed services that store authenticated (also called self-verifying) data. On the other hand, the RQS in which $\boldsymbol{QC_1} = \emptyset$ and $\boldsymbol{QC_2} = \boldsymbol{RQS}$ is a *masking quorum system* in the sense of [22]. These systems have been used to build resilient distributed services that store unauthenticated data.

So far, in examples 2-4, we considered refined quorum systems in which $\boldsymbol{QC_1} = \emptyset$. In the rest of the paper, we study the more general case where $\boldsymbol{QC_1} \neq \emptyset$. This is the case where our refined quorum systems capture both the resilience and the best-case complexity dimensions of distributed algorithms.

Example 5. Consider the case of a RQS where $\emptyset \neq \boldsymbol{QC_1} = \boldsymbol{QC_2}$. Such a RQS was implicitly used in [20], for the specific case $\boldsymbol{B} = \emptyset$, to devise a consensus algorithm that tolerates asynchronous periods and a threshold t of process (crash) failures, yet expedites decisions in best-case scenarios. In fact, although not used in the algorithm, the idea of a *fast* quorum (class 1 quorum in our terminology) was used to explain the algorithm. In the special case of an adversary $\boldsymbol{B_k}$, where (a) $\boldsymbol{RQS} = \boldsymbol{Q_t}$, and (b) $\boldsymbol{QC_1} = \boldsymbol{QC_2} = \boldsymbol{Q_q}$ ($q \leq t$), Property 2 is satisfied if $|S| > 2q + t + 2k$ and Property 1 is satisfied if $|S| > 2t + k$. These inequalities correspond to Lamport's lower bounds for "asynchronous" consensus [19]. The special case $k = q = t$ (i.e., where $\boldsymbol{QC_1} = \boldsymbol{RQS}$) corresponds to the refined quorum system implicitly used in [1, 23], also with the goal of boosting consensus latency.

Example 6. Even more interesting is the general case where $\emptyset \neq \boldsymbol{QC_1} \neq \boldsymbol{QC_2} \subseteq \boldsymbol{RQS}$ (e.g., Fig. 3), especially when $\boldsymbol{RQS}$, $\boldsymbol{QC_1}$ and the adversary are defined as in Example 4, $\boldsymbol{QC_2} = \boldsymbol{Q_r}$, and $0 \leq q < r \leq t$. In other words, each quorum contains all but at most t processes, while class 1 (resp., class 2) quorums contain all but at most q (resp., r) elements. $\boldsymbol{RQS}$ satisfies (i) Property 1 if $|S| > 2t + k$, (ii) Property 2 if $|S| > t + 2k + 2q$, and (iii) Property 3 if $|S| > t + r + k + min(k, q)$, i.e., $\boldsymbol{RQS}$ is a refined quorum system if $|S| > t + k + max(t, k + 2q, r + min(k, q))$. This quorum system was implicitly used in [7, 11], and later in [29].

3. CONCLUDING REMARKS

This paper introduces the notion of *refined quorum systems* (RQS) and argues that this is a useful notion to reason about optimally resilient and efficient distributed object implementations. We show [13] that RQS are necessary and sufficient (or, in a sense, minimal) for implementing an important class of *atomic* objects, namely atomic storage and consensus. This minimality holds when we indeed require atomicity and do not rely on authentication primitives to cope with Byzantine failures in best-case executions.

In the above context, and roughly speaking, denoting the best possible latency of an object implementation by l_1[2] (i.e., 1 round in the case of storage, or 2 message delays in the case of (Byzantine and asynchronous [19]) consensus), and by l_2 and l_3, incrementally, the next best possible latencies according to the corresponding metric, we proposed RQS-based object implementations that achieve a latency of l_i whenever a quorum of class i is available and best-case conditions (namely, synchrony and no-contention) are met. Since Property 1 of RQS (defined on class 3 quorums) is anyway necessary for any resilient implementation of distributed storage and consensus in an asynchronous environment, there is no need for refining quorums further. It might be important to notice here that the very notion of refined quorum system helps highlight the information structure of optimally resilient and best-case efficient atomic object implementations (at least those implementing the abstractions of atomic storage or consensus). Basically, these implementations inherently have three rounds together with a backup subprotocol in case of asynchrony or contention.

Our study opens at least two research directions. First, it would be interesting to carefully look into non-atomic semantics, e.g., regular or safe storage [17]. Recent results (in the threshold-based context) suggest that some (yet not all) properties of our refined quorum system are necessary and sufficient even for achieving optimal best-case complexity of weaker object implementations. Namely [2, 12] suggest that Properties 1 and 3a of RQS are necessary and sufficient for non-atomic best-case efficient storage implementations. These properties correspond to the special case

[2]This can be measured by the best possible latency in synchronous, uncontended and failure-free situations.

of RQS where $\boldsymbol{QC_1} = \emptyset$. Second, it would also be interesting to look into atomic object implementations that use data authentication in best-case executions. The lower bounds of [19], stated in the threshold-based context, suggest that Properties 1 and 2 are necessary and sufficient for best-case efficient and optimally resilient consensus implementations regardless of whether authentication is used in the best-case. These properties correspond to the special case of RQS where $\boldsymbol{QC_2} = \boldsymbol{QC_1}$. This suggests a general RQS-based framework for optimally efficient and resilient distributed objects, parameterized by the use of authentication and the desire for atomicity.

4. REFERENCES

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and Jay J. Wylie. Fault-scalable byzantine fault-tolerant services. In *Proceedings of the 20th ACM symposium on Operating systems principles*, pages 59–74, New York, NY, USA, 2005. ACM Press.

[2] I. Abraham, G. V. Chockler, I. Keidar, and D. Malkhi. Byzantine disk paxos: optimal resilience with Byzantine shared memory. *Distributed Computing*, 18(5):387–408, 2006.

[3] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.

[4] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation*, New Orleans, USA, February 1999.

[5] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementations*, Seattle, Washington, November 2006.

[6] P. Dutta, R. Guerraoui, R. R. Levy, and A. Chakraborty. How fast can a distributed atomic read be? In *Proceedings of the 23rd annual ACM symposium on Principles of distributed computing*, pages 236–245, New York, NY, USA, 2004. ACM Press.

[7] P. Dutta, R. Guerraoui, and M. Vukolić. Best-case complexity of asynchronous Byzantine consensus. Technical Report 200499, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, 2005.

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

[9] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM symposium on Operating systems principles*, pages 150–162, New York, NY, USA, 1979. ACM Press.

[10] G. Goodson, J. Wylie, G. Ganger, and M. Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 135–144, 2004.

[11] R. Guerraoui, R. R. Levy, and M. Vukolić. Lucky read/write access to robust atomic storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 125–136, Washington, DC, USA, 2006. IEEE Computer Society.

[12] R. Guerraoui and M. Vukolić. How Fast Can a Very Robust Read Be? In *25th ACM Symposium on Principles of Distributed Computing*, 2006.

[13] R. Guerraoui and M. Vukolić. Refined quorum systems. Technical Report LPD-REPORT-2007-002, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, February 2007.

[14] M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems*, 13(1):124–149, January 1991.

[15] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In *Proceedings of the 16th annual ACM symposium on Principles of distributed computing*, pages 25–34, New York, NY, USA, 1997. ACM Press.

[16] P. Jayanti, T. D. Chandra, and S. Toueg. Fault-tolerant wait-free shared objects. *Journal of the ACM*, 45(3):451–500, 1998.

[17] L. Lamport. On interprocess communication. *Distributed computing*, 1(1):77–101, May 1986.

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

[19] L. Lamport. Lower bounds for asynchronous consensus. In *Future Directions in Distributed Computing*, Springer Verlag (LNCS), pages 22–23, 2003.

[20] L. Lamport. Fast Paxos. *Distributed Computing*, 19(2):79–103, 2006.

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

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

[23] J.-P. Martin and L. Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[24] J.-P. Martin, L. Alvisi, and M. Dahlin. Minimal Byzantine storage. In *Proceedings of the 16th International Conference on Distributed Computing*, pages 311–325. Springer-Verlag, 2002.

[25] M. Naor and A. Wool. The load, capacity and availability of quorum systems. In *Proceedings of the 35th IEEE Symposium on Foundations of Computer Science*, pages 214–225, 1994.

[26] H. V. Ramasamy and C. Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In *Proceedings of the 9th International Conference on Principles of Distributed Systems*, Lecture Notes in Computer Science, pages 88–102, December 2005.

[27] Y. Saito, S. Frolund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. *SIGOPS Oper. Syst. Rev.*, 38(5):48–58, 2004.

[28] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2):180–209, 1979.

[29] P. Zieliński. Optimistically terminating consensus. Technical Report UCAM-CL-TR-668, Cambridge University, Cambridge, UK, June 2006.