Byzantine Quorum Systems*

Dahlia Malkhi Michael Reiter

AT&T Labs—Research, Florham Park, NJ USA
{dalia,reiter}@research.att.com

October 16, 1998

Abstract

Quorum systems are well-known tools for ensuring the consistency and availability of replicated data despite the benign failure of data repositories. In this paper we consider the arbitrary (Byzantine) failure of data repositories and present the first study of quorum system requirements and constructions that ensure data availability and consistency despite these failures. We also consider the load associated with our quorum systems, i.e., the minimal access probability of the busiest server. For services subject to arbitrary failures, we demonstrate quorum systems over n servers with a load of $O(\frac{1}{\sqrt{n}})$, thus meeting the lower bound on load for benignly fault-tolerant quorum systems. We explore several variations of our quorum systems and extend our constructions to cope with arbitrary client failures.

1 Introduction

A well known way to enhance the availability and efficiency of replicated data is by using *quorums*. A quorum system for a universe of data servers is a collection of subsets of servers, each pair of which intersect. Intuitively, each quorum can operate on behalf of the system, thus increasing its availability and performance, while the intersection property guarantees that operations done on distinct quorums preserve consistency.

In this paper we consider the arbitrary (Byzantine) failure of clients and servers, and initiate the study of quorum systems in this model. Intuitively, a quorum system tolerant of Byzantine failures is a collection of subsets of servers, each pair of which intersect in a set containing sufficiently many *correct* servers to guarantee consistency of the replicated data as seen by clients. We provide the following contributions.

1. We define the class of *masking quorum systems*, with which data can be consistently replicated in a way that is resilient to the arbitrary failure of data repositories. We show necessary and sufficient conditions for the existence of masking quorum systems under different failure assumptions, and present several example constructions of such systems.

2. We explore two variations of masking quorum systems. The first, called *dissemination quorum systems*, is suited for services that receive and distribute *self-verifying* information from correct clients (e.g., digitally signed values) that faulty servers can fail to redistribute but cannot undetectably alter. The second variation, called *opaque masking quorum systems*, is similar to

*preprint of paper to appear in the Journal of Distributed Computing, 11(4), 1998.

regular masking quorums in that it makes no assumption of self-verifying data, but it differs in that clients do not need to know the failure scenarios for which the service was designed. This somewhat simplifies the protocol by which clients access the replicated data and, in the case that failures are maliciously induced, reveals less information to clients that could guide an attack attempting to compromise the system.

3. We explore the *load* of each type of quorum system, where the load of a quorum system is the minimal access probability of the busiest server, minimizing over all strategies for picking quorums. We present a masking quorum system with the property that its load over a total of n servers is $O(\frac{1}{\sqrt{n}})$, thereby meeting the lower bound for the load of benignly-fault-tolerant quorum systems. For opaque masking quorum systems, we prove a lower bound of $\frac{1}{2}$ on the load, and present a construction that meets this lower bound and proves it tight.

4. For services that use masking quorums (opaque or not), we show how to deal with faulty clients in addition to faulty servers. The primary challenge raised by client failures is that there is no guarantee that clients will update quorums according to any specified protocol. Thus, a faulty client could leave the replicated data in an inconsistent and irrecoverable state. We develop an update protocol, by which clients update the replicated data, that prevents clients from leaving the data in an inconsistent state. The protocol has the desirable property that it involves only the quorum at which an access is attempted, while providing system-wide consistency properties.

Our quorum systems, if used in conjunction with appropriate protocols and synchronization mechanisms, can be used to implement a wide range of data semantics. In this paper, however, we choose to demonstrate a variable supporting read and write operations with relatively weak semantics, in order to maintain focus on our quorum constructions. These semantics imply a *safe* variable [24], which a set of correct clients can use to build other abstractions, e.g., atomic, multi-writer multi-reader registers [24, 21, 25], concurrent timestamp systems [12, 19], l-exclusion [11, 2], and atomic snapshot scan [1, 5].

Our quorum systems can be used for building other protocols in addition to shared read/write register emulation. For example, in an ongoing effort [30], we use Byzantine quorum systems in constructing a large-scale, survivable service supporting persistent data abstractions such as consensus objects [29], locks and files. In addition, in Section 6, we demonstrate how masking quorum systems can be used to guarantee consistency and completion of updates, even those executed by faulty clients.

The rest of this paper is structured as follows. We begin in Section 2 with a description of related work. In Section 3 we present our system model and definitions. We present quorum systems for the replication of arbitrary data subject to arbitrary server failures in Section 4, and in Section 5 we present two variations of these systems. We then detail an access protocol for replicated services that tolerate faulty clients in addition to faulty servers in Section 6. We conclude in Section 7.

2 Related work

Our work was influenced by the substantial body of literature on quorum systems for benign failures and applications that make use of them, e.g., [15, 42, 26, 14, 17, 13, 9, 4, 35]. In particular, our grid construction of Section 4 was influenced by grid-like constructions for benign failures (e.g., [9]), and we borrow our definition of *load* from [35].

Quorum systems have been previously employed in the implementation of security mechanisms. Naor and Wool [36] described methods to construct an access-control service using quorums. Their constructions use cryptographic techniques to ensure that out-of-date (but correct) servers cannot grant access to unauthorized users. Agrawal and El Abbadi [3] and Mukkamala [34] considered the confidentiality of replicated data despite the disclosure of the contents of a threshold of the (otherwise correct) repositories. Their constructions used quorums with increased intersection, combined with Rabin's dispersal scheme [37], to enhance the confidentiality and availability of the data despite some servers crashing or their contents being observed. Our work differs from all of the above by considering arbitrarily faulty servers, and accommodating failure scenarios beyond a simple threshold of servers.

Herlihy and Tygar [18] applied quorums with increased intersection to the problem of protecting the confidentiality and integrity of replicated data against a threshold of arbitrarily faulty servers. In their constructions, replicated data is stored encrypted under a key that is shared among the servers using a threshold secret-sharing scheme [40], and each client accesses a threshold number of servers to reconstruct the key prior to performing (encrypted) reads and writes. This construction exhibits one approach to make replicated data self-verifying via encryption, and thus the quorum system they develop is a special case of our dissemination quorum systems, i.e., for a threshold of faulty servers.

Since the initial conference publication of this work [28], several works that build upon its contributions have appeared. A subsequent paper [31] is devoted to constructions of masking quorum systems for the special case of a threshold of faulty servers. Bazzi [6] explored a variation of our quorum systems for synchronous systems. Probabilistic constructions for dissemination and masking quorum systems are explored in [32] and [33], respectively. A practical effort for building a large-scale survivable data repository using Byzantine quorums is described in [29], and the construction of a survivable consensus object in this context is described in [30].

3 Preliminaries

3.1 System model

We assume a *universe* U of servers, $|U| = n$, and an arbitrary number of clients that are distinct from the servers. A *quorum system* $\mathcal{Q} \subseteq 2^U$ is a non-empty set of subsets of U, every pair of which intersect. Each $Q \in \mathcal{Q}$ is called a *quorum.*

Servers (and clients) that obey their specifications are *correct.* A *faulty* server, however, may deviate from its specification arbitrarily. A *fail-prone system* $\mathcal{B} \subseteq 2^U$ is a non-empty set of subsets of U, none of which is contained in another, such that some $B \in \mathcal{B}$ contains all the faulty servers. The fail-prone system represents an assumption characterizing the failure scenarios that can occur, and could express typical assumptions that up to a threshold of servers fail (e.g., the sets $B_1, \ldots, B_k$ could be all sets of f servers), but it also generalizes to allow less uniform assumptions. For example, servers in physical proximity to each other or in the same administrative domain may exhibit correlated probabilities of being captured, or servers with identical hardware and software platforms may have correlated probabilities of electronic penetration. By exploiting such correlations (i.e., knowledge of the collection $\mathcal{B}$), we can design quorum systems that more effectively mask faulty servers.

In the remainder of this section, and throughout Sections 4 and 5, we assume that clients behave correctly. In Section 6 we will relax this assumption (and will be explicit when we do so).

We assume that any two processes (clients or servers) can communicate over a point-to-point channel. If both endpoints of the channel are correct, then this channel is both authenticated and reliable. That is, a correct process receives a message from another correct process if and only if the other correct process sent it. However, we do *not* assume known bounds on message transmission times; i.e., communication is asynchronous.

3.2 Access protocol

We consider a problem in which the clients perform read and write operations on a variable x that is replicated at each server in the universe U. A copy of the variable x is stored at each server, along with a timestamp value t. Timestamps are assigned by a client to each replica of the variable when the client writes the replica. Our protocols require that different clients choose different timestamps, and thus each client c chooses its timestamps from some globally-known set T_c that does not intersect $T_{c'}$ for any other client c'. The timestamps in T_c can be formed, e.g., as integers appended with the name of c in the low-order bits. The read and write operations are implemented as follows.

Write: For a client c to write the value v, it queries servers to obtain a set of timestamps $A = \{<t_u>\}_{u \in Q}$ for some quorum Q; chooses a timestamp $t \in T_c$ greater than the highest timestamp value in A and greater than any timestamp it has chosen in the past; and sends the update $<v, t>$ to servers until it has received an acknowledgement for this update from every server in some quorum Q'.

Read: For a client to read x, it queries servers to obtain a set of value/timestamp pairs $A = \{<v_u, t_u>\}_{u \in Q}$ for some quorum Q. The client then applies a deterministic function *Result*() to A to obtain the result *Result*(A) of the read operation.

In the case of a write operation, each server updates its local variable and timestamp to the received values $<v, t>$ only if t is greater than the timestamp currently associated with the variable. In any case, it returns an acknowledgement to the client.

Two points about this description deserve further discussion. First, the nature of the quorums Q and the function *Result*() are intentionally left unspecified; further clarification of these are the point of this paper. Second, read and write operations need to exchange messages with a full quorum of servers. For example, the read operation requires a client to obtain a set A containing value/timestamp pairs from *every* server in some quorum Q. This requirement stems from our lack of synchrony assumptions on the network: in general, the only way that a client can know that it has accessed every *correct* server in a quorum is to access every server in the quorum. Our framework guarantees the availability of a quorum at any moment, and thus by attempting the operation at multiple quorums, a client can eventually make progress. In some cases, the client can achieve progress by incrementally accessing servers until it obtains responses from a quorum of them.

In Sections 4 and 5, we will argue the correctness of the above protocol—instantiated with quorums and a *Result*() function that we will define—according to the following semantics; a more formal treatment of these concepts can be found in [24]. We say that a read operation *begins* when the client initiates the operation and *ends* when the client obtains the read value; an operation to write value v with timestamp t *begins* when the client initiates it and *ends* when all correct servers in some quorum have received the update $<v, t>$. An operation op_1 *precedes* an operation op_2 if op_1 ends before op_2 begins (in real time). If op_1 does not precede op_2 and op_2 does not precede op_1, then they are called *concurrent*. Given a set of operations, a *serialization* of those operations is a total ordering on them that extends the precedence ordering among them. Then, for the above protocol to be correct, we require that any read that is concurrent with no writes returns the last value written in some serialization of the preceding writes. This will immediately imply *safe* variable semantics [24].

3.3 Load

A measure of the inherent performance of a quorum system is its *load* [35], defined as follows: Given a quorum system $\mathcal{Q}$, an *access strategy* w is a probability distribution on the elements of $\mathcal{Q}$; i.e., $\sum_{Q\in\mathcal{Q}} w(Q) = 1$. $w(Q)$ is the probability that quorum Q will be chosen when the service is accessed. Load is then defined as follows:

Definition 3.1 Let a strategy w be given for a quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ over a universe U. For an element $u \in U$, the load induced by w on u is $l_w(u) = \sum_{Q_i \ni u} w(Q_i)$. The load induced by a strategy w on a quorum system $\mathcal{Q}$ is

$$L_w(\mathcal{Q}) = \max_{u\in U}\{l_w(u)\}.$$

The *system load* (or just *load*) on a quorum system $\mathcal{Q}$ is

$$L(\mathcal{Q}) = \min_{w}\{L_w(\mathcal{Q})\},$$

where the minimum is taken over all strategies. $\Box$

We reiterate that the load is a best case definition. The load of the quorum system will be achieved only if an optimal access strategy is used, and only in the case that no failures occur. A strength of this definition is that load is a property of a quorum system, and not of the protocol using it. A comparison of the definition of load to other seemingly plausible definitions is given in [35].

4 Masking quorum systems

In this section we introduce *masking quorum systems*, which can be used to mask the arbitrarily faulty behavior of data repositories. To motivate our definition, suppose that the replicated variable x is written with quorum Q_1, and that subsequently x is read using quorum Q_2. If B is the set of arbitrarily faulty servers, then the following is obtained by reading from Q_2: the correct value for x is obtained from each server in $(Q_1 \cap Q_2) \setminus B$ (see Figure 1); out-of-date values are obtained from $Q_2 \setminus (Q_1 \cup B)$; and arbitrary values are obtained from $Q_2 \cap B$. In order for the client to obtain the correct value, the client must be able to identify the most up-to-date value/timestamp pair as one returned by a set of servers that could not all be faulty. This yields requirement M-Consistency below. In addition, since communication is asynchronous and thus accurate failure detection is not possible, in order for a client to know it completes an operation with all the correct servers of some quorum, it must be able to obtain responses from a full quorum. Therefore, for availability we require that there be no set of faulty servers that intersects all quorums.

Definition 4.1 A quorum system $\mathcal{Q}$ is a *masking quorum system* for a fail-prone system $\mathcal{B}$ if the following properties are satisfied.

M-Consistency: $\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B_1, B_2 \in \mathcal{B}: \quad (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$

M-Availability: $\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q}: \quad B \cap Q = \emptyset$

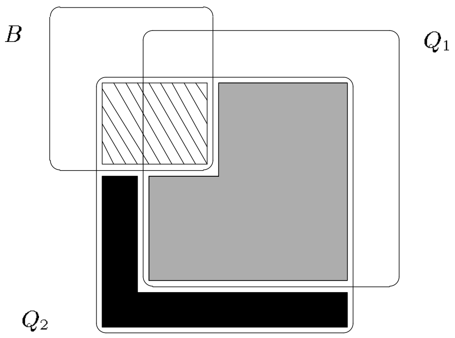

Figure 1: Reading from a masking quorum Q_2

□

For example, in the case that at most f servers can fail, M-Consistency guarantees that every pair of quorums intersect in at least $2f+1$ elements, and thus in $f+1$ correct ones. If a read operation accepts only a value returned by at least $f+1$ servers, then any accepted value was returned by at least one correct server.

More generally, the masking quorum system requirements enable a client to obtain the correct answer from the service despite the Byzantine failure of any fail-prone set. The write operation is implemented as described in Section 3. To obtain the correct value of x from a read operation, the client reads a set of value/timestamp pairs from a quorum Q, discards values that are returned from any $B' \in \mathcal{B}$ or subsets thereof, and chooses among the remaining values the one with the highest timestamp. This guarantees correctness of the returned value/timestamp pair, which was received from some set $B^+ \subseteq Q$ of servers, where B^+ is not contained in any $B' \in \mathcal{B}$ and therefore must contain at least one correct server. Furthermore, it is easy to see that if the most recent write has completed in quorum Q', then all of the servers in $Q \cap Q' \setminus B$ will return this most up-to-date value, and since by definition $Q \cap Q' \setminus B$ is not contained in any $B' \in \mathcal{B}$, this value will be returned by the read operation. The read operation is thus as follows:

Read: For a client to read a variable x, it queries servers to obtain a set of value/timestamp pairs $A = \{<v_u, t_u>\}_{u \in Q}$ for some quorum Q. The client computes the set

$$A' = \{<v,t> \; : \; \exists B^+ \subseteq Q \; [\; \forall B \in \mathcal{B} \; [B^+ \not\subseteq B] \wedge \\ \forall u \in B^+ \; [v_u = v \wedge t_u = t] \;]\}.$$

The client then chooses the pair $<v,t>$ in A' with the highest timestamp, and chooses v as the result of the read operation; if A' is empty, the client returns $\perp$ (a null value, which indicates that the read failed).

Lemma 4.2 A read operation that is concurrent with no write operations returns the value written by the last preceding write operation in some serialization of all preceding write operations.

Proof. Let W denote the set of write operations preceding the read. The read operation will return the value written in the write operation in W with the highest timestamp, since, by the construction of masking quorum systems, this value/timestamp pair will appear in A' and will have the highest timestamp in A' (any pair with a higher timestamp will be returned only by servers in some $B \in \mathcal{B}$). So, it suffices to argue that there is a serialization of the writes in W in which this write operation appears last, or in other words, that this write operation precedes no other write operation in W. This is immediate, however, as if it did precede another write operation in W, that write operation would have a higher timestamp. $\Box$

This lemma implies that the protocol above implements a *multi-writer multi-reader safe* variable [24]. A failure value ($\perp$) may be returned when some write overlaps a read operation. Nevertheless, from safe variables multi-writer multi-reader atomic variables can be built using well-known constructions [24, 21, 25].

A necessary and sufficient condition for the existence of a masking quorum system (and a construction for one, if it exists) for any given fail-prone system $\mathcal{B}$ is given in the following theorem:

Theorem 4.3 Let $\mathcal{B}$ be a fail-prone system for a universe U. Then there exists a masking quorum system for $\mathcal{B}$ iff $\mathcal{Q} = \{U \setminus B : B \in \mathcal{B}\}$ is a masking quorum system for $\mathcal{B}$.

Proof. Obviously, if $\mathcal{Q}$ is a masking quorum system for $\mathcal{B}$, then one exists. To show the converse, assume that $\mathcal{Q}$ is not a masking quorum. Since M-Availability holds in $\mathcal{Q}$ by construction, there exist $Q_1, Q_2 \in \mathcal{Q}$ and $B', B'' \in \mathcal{B}$, such that $(Q_1 \cap Q_2) \setminus B' \subseteq B''$. Let $B_1 = U \setminus Q_1$ and $B_2 = U \setminus Q_2$. By the construction of $\mathcal{Q}$, we know that $B_1, B_2 \in \mathcal{B}$. By M-Availability, any masking quorum system for $\mathcal{B}$ must contain quorums $Q'_1 \subseteq Q_1$, $Q'_2 \subseteq Q_2$. However, for any such Q'_1, Q'_2, it is the case that $(Q'_1 \cap Q'_2) \setminus B' \subseteq (Q_1 \cap Q_2) \setminus B' \subseteq B''$, violating M-Consistency. Therefore, there does not exist a masking quorum system for $\mathcal{B}$ under the assumption that $\mathcal{Q}$ is not a masking quorum system for $\mathcal{B}$. $\Box$

Corollary 4.4 Let $\mathcal{B}$ be a fail-prone system for a universe U. Then there exists a masking quorum system for $\mathcal{B}$ iff for all $B_1, B_2, B_3, B_4 \in \mathcal{B}$, $U \not\subseteq B_1 \cup B_2 \cup B_3 \cup B_4$. In particular, suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$. Then, there exists a masking quorum system for $\mathcal{B}$ iff $n > 4f$.

Proof. By Theorem 4.3, there is a masking quorum for $\mathcal{B}$ iff $\mathcal{Q} = \{U \setminus B : B \in \mathcal{B}\}$ is a masking quorum for $\mathcal{B}$. By construction, $\mathcal{Q}$ is a masking quorum iff M-Consistency holds for $\mathcal{Q}$, i.e., iff for all $B_1, B_2, B_3, B_4 \in \mathcal{B}$:

$$\begin{aligned} & ((U \setminus B_1) \cap (U \setminus B_2)) \setminus B_3 \not\subseteq B_4 \\ \iff & U \setminus (B_1 \cup B_2) \not\subseteq B_3 \cup B_4 \\ \iff & U \not\subseteq B_1 \cup B_2 \cup B_3 \cup B_4. \end{aligned}$$

$\Box$

The existence criterion for masking quorum systems identified by Theorem 4.3 characterizes all possible masking systems for the fail-prone system $\mathcal{B}$. In particular, the system $\mathcal{Q}$ in Theorem 4.3 is *dominated* (in the sense of [14]) by any other masking quorum system $\mathcal{Q}'$ for $\mathcal{B}$, in that for every $Q \in \mathcal{Q}$ there must exist $Q' \in \mathcal{Q}'$ such that $Q' \subseteq Q$. While this provides a characterization of masking quorum systems for any fail-prone system $\mathcal{B}$, it does not help in constructing ones to meet any specific requirements. Garcia-Molina and Barbara [14] present techniques for enumerating a certain class of (non-Byzantine) quorum systems. Their methods are not directly applicable for enumerating masking quorum systems, and we leave as an open research topic the question of efficiently mechanizing masking quorum generation. A separate paper [31] provides constructions that are optimal in load and various availability measures for any threshold failure assumption up to the maximum of $n/4$.

The following theorem was proved in [35] for benign-failure quorum systems, and holds a fortiori for masking quorums (as a result of M-Consistency). Let $c(\mathcal{Q})$ denote the size of the smallest quorum of $\mathcal{Q}$.

Theorem 4.5 [35] If $\mathcal{Q}$ is a quorum system over a universe of n elements, then $L(\mathcal{Q}) \geq \max\{\frac{1}{c(\mathcal{Q})}, \frac{c(\mathcal{Q})}{n}\}$, and thus, $L(\mathcal{Q}) \geq \frac{1}{\sqrt{n}}$.

Below we give several examples of masking quorum systems and describe their properties.

Example 4.6 (*f-masking*) Suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $n > 4f$. Note that this corresponds to the usual threshold assumption that up to f servers may fail. Then, the quorum system $\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{n+2f+1}{2} \rceil\}$ is a masking quorum system for $\mathcal{B}$. M-Consistency is satisfied because any $Q_1, Q_2 \in \mathcal{Q}$ will intersect in at least $2f+1$ elements. M-Availability holds because $\lceil \frac{n+2f+1}{2} \rceil \leq n - f$. A strategy that assigns equal probability to each quorum induces a load of $\frac{1}{n}\lceil \frac{n+2f+1}{2} \rceil$ on the system. By Theorem 4.5, this load is in fact the load of the system. □

The following example is interesting since its load decreases as a function of n, and since it demonstrates a method for ensuring system-wide consistency in the face of Byzantine failures while requiring the involvement of fewer than a majority of the correct servers. These advantages are dramatic when n is sufficiently large, e.g., hundreds of servers.

Example 4.7 (*Grid quorums*) Suppose that the universe of servers is of size $n = k^2$ for some integer k and that $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $3f + 1 \leq \sqrt{n}$. Arrange the universe into a $\sqrt{n} \times \sqrt{n}$ grid, as shown in Figure 2. Denote the rows and columns of the grid by R_i and C_i, respectively, where $1 \leq i \leq \sqrt{n}$. Then, the quorum system

$$\mathcal{Q} = \left\{ C_j \cup \bigcup_{i \in I} R_i : I, \{j\} \subseteq \{1 \ldots \sqrt{n}\}, |I| = 2f+1 \right\}$$

is a masking quorum system for $\mathcal{B}$. M-Consistency holds since every pair of quorums intersect in at least $2f+1$ elements (the column of one quorum intersects the $2f+1$ rows of the other), and M-Availability holds since for any choice of f faulty elements in the grid, $2f+1$ full rows and a column remain available. A strategy that assigns equal probability to each quorum induces a load of $\frac{(2f+2)\sqrt{n}-(2f+1)}{n}$, and again by Theorem 4.5, this is the load of the system. □

Note that by choosing $\mathcal{B} = \{\emptyset\}$ (i.e., $f = 0$) in the example above, the resulting construction has a load of $O(\frac{1}{\sqrt{n}})$, which asymptotically meets the bounds given in Theorem 4.5. In general, however, this construction yields a load of $O(\frac{f}{\sqrt{n}})$, which is not optimal: Malkhi et al. [31] show a lower bound of $\sqrt{\frac{2f+1}{n}}$ on the load of any masking quorum system for $\mathcal{B} = \{B \subseteq U : |B| = f\}$, and provide a construction whose load matches that bound.

Example 4.8 (*Partition*) Suppose that $\mathcal{B} = \{B_1, \ldots, B_m\}$, $m > 4$, is a partition of U where $B_i \neq \emptyset$ for all i, $1 \leq i \leq m$. This choice of $\mathcal{B}$ could arise, for example, in a wide area network composed of multiple local clusters, each consisting of some B_i, and expresses the assumption that at any time, at most one cluster is faulty. Then, any collection of nonempty sets $\hat{B}_i \subseteq B_i$, $1 \leq i \leq m$, can

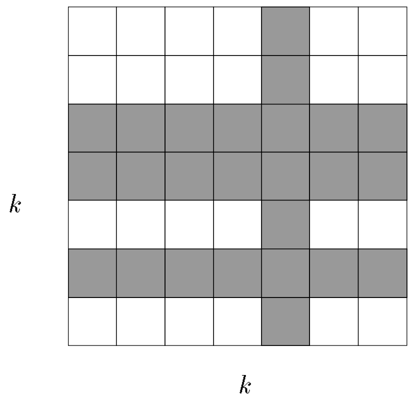

Figure 2: Grid construction, $k \times k = n$, $f = 1$ (one quorum shaded).

be thought of as 'super-elements' in a universe of size m, with a threshold assumption $f = 1$ (see Figure 3). Therefore, the following is a masking quorum system for $\mathcal{B}$:

$$\mathcal{Q} = \left\{ \bigcup_{i \in I} \hat{B}_i \ : \ I \subseteq \{1, \ldots, m\}, |I| = \lceil \tfrac{m+3}{2} \rceil \right\}$$

M-Consistency is satisfied because the intersection of any two quorums contains elements from at least three sets in $\mathcal{B}$. M-Availability holds since there is no $B \in \mathcal{B}$ that intersects all quorums. A strategy that assigns equal probability to each quorum induces a load of $\frac{1}{m}\lceil \frac{m+3}{2} \rceil$ on the system regardless of the size of each $\hat{B}_i$, and again Theorem 4.5 implies that this is the load of the system.

If $m = k^2$ for some k, then a more efficient construction can be achieved by forming the grid construction from Example 4.8 on the 'super elements' $\{\hat{B}_i\}$, achieving a load of $\frac{4\sqrt{m}-3}{m}$. $\square$

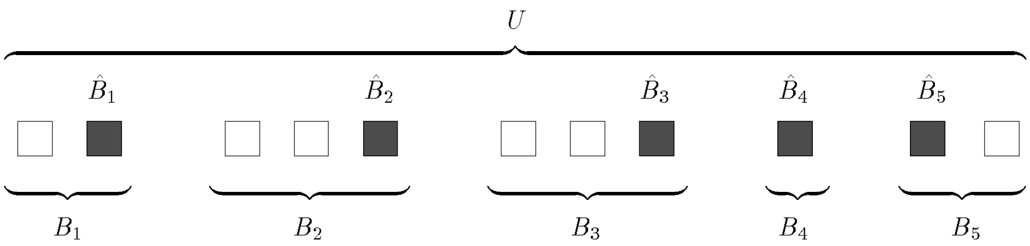

Figure 3: Partition $\{B_1, B_2, B_3, B_4, B_5\}$, $\hat{B}_i$'s shaded.

5 Variations

5.1 Dissemination quorum systems

As a special case of services that can employ quorums in a Byzantine environment, we now consider applications in which the service is a repository for self-verifying information, i.e., information that only clients can create and to which clients can detect any attempted modification by a faulty server. A natural example is a database of *public key certificates* as found in many public key distribution systems (e.g., [10, 41, 23]). In its simplest form, a public key certificate is a structure containing a name for a user and a public key, and represents the assertion that the indicated public key can be used to authenticate messages from the indicated user. This structure is digitally signed (e.g., [39]) by a *certification authority* so that anyone with the public key of this authority can verify this assertion and, providing it trusts the authority, use the indicated public key to authenticate the indicated user. Due to this signature, it is not possible for a faulty server to undetectably modify a certificate it stores. However, a faulty server *can* undetectably suppress a change from propagating to clients, simply by ignoring an update from a certification authority. This could have the effect, e.g., of suppressing the revocation of a key that has been compromised.

As can be expected, the use of digital signatures to verify data decreases the cost of accessing replicated data. To support such a service, we employ a *dissemination quorum system*, which has weaker requirements than masking quorums, but which nevertheless ensures that in applications like those above, self-verifying writes will be propagated to all subsequent read operations despite the arbitrary failure of some servers. To achieve this, it suffices for the intersection of every two quorums to not be contained in any set of potentially faulty servers (so that a written value can propagate to a read). This leads to requirement D-Consistency below. And, supposing that operations are required to continue in the face of failures, then due to the lack of accurate failure detection, there should be quorums that a faulty set cannot disable; this yields requirement D-Availability below.

Definition 5.1 A quorum system $\mathcal{Q}$ is a *dissemination quorum system* for a fail-prone system $\mathcal{B}$ if the following properties are satisfied.

D-Consistency: $\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B \in \mathcal{B}: \quad Q_1 \cap Q_2 \not\subseteq B$

D-Availability: $\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q}: \quad B \cap Q = \emptyset$

□

A dissemination quorum system will suffice for propagating self-verifying information as in the application described above. The write operation is implemented as described in Section 3, and the read operation becomes:

Read: For a client to read a variable x, it queries servers to obtain a set of value/timestamp pairs $A = \{<v_u, t_u>\}_{u \in Q}$ for some quorum Q. The client then discards those pairs that are not verifiable (e.g., using an appropriate digital signature verification algorithm) and chooses from the remaining pairs the pair $<v, t>$ with the largest timestamp. v is the result of the read operation.

It is important to note that timestamps must be included as part of the self-verifying information, so they cannot be undetectably altered by faulty servers. In the case of the application described above, existing standards for public key certificates (e.g., [10]) already require a real-time timestamp in the certificate.

The following lemma proves correctness of the above protocol using dissemination quorum systems. The proof is almost identical to that for masking quorum systems.

Lemma 5.2 A read operation that is concurrent with no write operations returns the value written by the last preceding write operation in some serialization of all preceding write operations.

Due to the assumption of self-verifying data, we can also prove in this case the following property.

Lemma 5.3 A read operation that is concurrent with one or more write operations returns either the value written by the last preceding write operation in some serialization of all preceding write operations, or any of the values being written in the concurrent write operations.

The above lemmata imply that the protocol above implements a *regular* variable [24]. Theorems analogous to the ones given for masking quorum systems above are easily derived for dissemination quorums. Below, we list these results without proof.

Theorem 5.4 Let $\mathcal{B}$ be a fail-prone system for a universe U. Then there exists a dissemination quorum system for $\mathcal{B}$ iff $\mathcal{Q} = \{U \setminus B : B \in \mathcal{B}\}$ is a dissemination quorum system for $\mathcal{B}$.

Corollary 5.5 Let $\mathcal{B}$ be a fail-prone system for a universe U. Then there exists a dissemination quorum system for $\mathcal{B}$ iff for all $B_1, B_2, B_3 \in \mathcal{B}$, $U \not\subseteq B_1 \cup B_2 \cup B_3$. In particular, suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$. Then, there exists a dissemination quorum system for $\mathcal{B}$ iff $n > 3f$.

Below, we provide several example constructions of dissemination quorum systems.

Example 5.6 (*f-dissemination*) Suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $n > 3f$. Note that this corresponds to the usual threshold assumption that up to f servers may fail. Then, the quorum system $\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{n+f+1}{2} \rceil\}$ is a dissemination quorum system for $\mathcal{B}$ with load $\frac{1}{n}\lceil \frac{n+f+1}{2} \rceil$. $\square$

Example 5.7 (*Grid*) Let the universe be arranged in a grid as in Example 4.8 above, and let $\mathcal{B} = \{B \subseteq U : |B| = f\}$, $2f + 1 \leq \sqrt{n}$. Then, the quorum system

$$\mathcal{Q} = \left\{ C_j \cup \bigcup_{i \in I} R_i : I, \{j\} \subseteq \{1 \ldots \sqrt{n}\}, |I| = f + 1 \right\}$$

is a dissemination quorum system for $\mathcal{B}$. The load of this system is $\frac{(f+2)\sqrt{n}-(f+1)}{n}$. $\square$

Example 5.8 (*Partition*) Suppose that $\mathcal{B} = \{B_1, \ldots, B_m\}$, $m > 3$, is a partition of U as in Figure 3. For any collection of nonempty sets $\hat{B}_i \subseteq B_i$, $1 \leq i \leq m$, the f-dissemination construction of Example 5.7 on the 'super-elements' $\hat{B}_i \subseteq B_i$ (as in Example 4.8) yields a dissemination quorum system with a load of $\frac{1}{m}\lceil \frac{m+2}{2} \rceil$. If $m = k^2$ for some k, the Grid construction of Example 5.8 achieves a load of $\frac{3\sqrt{m}-2}{m}$. $\square$

5.2 Opaque masking quorum systems

Masking quorums impose a requirement that clients know the fail-prone system $\mathcal{B}$, while there may be reasons that clients should not be required to know this. First, it somewhat complicates the client's read protocol, in particular, when no concise description of $\mathcal{B}$ exists. Second, by revealing the failure scenarios for which the system was designed, the system also reveals the failure scenarios to which it is vulnerable, which could be exploited by an attacker to guide an active attack against the system. By not revealing the fail-prone system to clients, and indeed giving each client only a small fraction of the possible quorums, the system can somewhat obscure (though perhaps not secure in any formal sense) the failure scenarios to which it is vulnerable, especially in the absence of client collusion.

In this section we describe one way to modify the masking quorum definition of Section 4 to be *opaque*, i.e., to eliminate the need for clients to know $\mathcal{B}$. In the absence of the client knowing

$\mathcal{B}$, the only method of which we are aware for the client to reduce a set of replies from servers to a single reply from the service is via *voting*, i.e., choosing the reply that occurs most often. In order for this reply to be the correct one, however, we must strengthen the requirements on our quorum systems. Specifically, suppose that the variable x is written with quorum Q_1, and that subsequently x is read with quorum Q_2. If B is the set of arbitrarily faulty servers, then $(Q_1 \cap Q_2) \setminus B$ is the set of correct servers that possess the latest value for x (see Figure 4). In order for the client to obtain this value by vote, this set must be larger than the set of faulty servers that are allowed to respond, i.e., $Q_2 \cap B$. Moreover, since these faulty servers can "team up" with the out-of-date but correct servers in an effort to suppress the write operation, the number of correct, up-to-date servers that reply must be no less than the number of faulty or out-of-date servers that can reply, i.e., $(Q_2 \cap B) \cup (Q_2 \setminus Q_1)$. Finally, to effectively mask failures by any $B \in \mathcal{B}$ in an asynchronous environment, we add the availability requirement (O-Availability).

Definition 5.9 A quorum system $\mathcal{Q}$ is an *opaque masking quorum system* for a fail-prone system $\mathcal{B}$ if the following properties are satisfied.

O-Consistency1: $\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B \in \mathcal{B}: \quad |(Q_1 \cap Q_2) \setminus B| \geq |(Q_2 \cap B) \cup (Q_2 \setminus Q_1)|$

O-Consistency2: $\forall Q_1, Q_2 \in \mathcal{Q}\ \forall B \in \mathcal{B}: \quad |(Q_1 \cap Q_2) \setminus B| > |Q_2 \cap B|$

O-Availability: $\forall B \in \mathcal{B}\ \exists Q \in \mathcal{Q}: \quad B \cap Q = \emptyset$

□

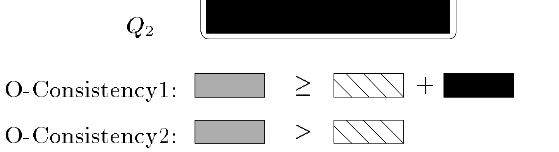

Figure 4: O-Consistency1 and O-Consistency2

Note that O-Consistency1 admits the possibility of equality in size between $(Q_1 \cap Q_2) \setminus B$ and $(Q_2 \cap B) \cup (Q_2 \setminus Q_1)$. Equality is sufficient since, in the case that the faulty servers "team up" with the correct but out-of-date servers in Q_2, the value returned from $(Q_1 \cap Q_2) \setminus B$ will have a higher timestamp than that returned by $(Q_2 \cap B) \cup (Q_2 \setminus Q_1)$. Therefore, in the case of a tie, a reader can choose the value with the higher timestamp. It is interesting to note that a strong inequality in O-Consistency1 would permit a correct implementation of a single-reader singer-writer safe variable that does not use timestamps (by taking the majority value in a read operation).

It is not difficult to verify that an opaque masking quorum system enables a client to obtain the correct answer from the service. The write operation is implemented as described in Section 1, and the read operation becomes:

Read: For a client to read a variable x, it queries servers to obtain a set of value/timestamp pairs $A = \{<v_u, t_u>\}_{u \in Q}$ for some quorum Q. The client chooses the pair $<v, t>$ that appears most often in A, and if there are multiple such pairs, the one with the highest timestamp. The value v is the result of the read operation.

Opaque masking quorum systems, combined with the access protocol described previously, provide the same semantics as regular masking quorum systems. The proof is almost identical to that for regular masking quorums.

Lemma 5.10 A read operation that is concurrent with no write operations returns the value written by the last preceding write operation in some serialization of all preceding write operations.

Below we give several examples of opaque masking quorum systems (or just "opaque quorum systems") and describe their properties.

Example 5.11 (*f-opaque*) Suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$ where $n \geq 5f$ and $f > 0$. Then, the quorum system $\mathcal{Q} = \{Q \subseteq U : |Q| = \lceil \frac{2n+2f}{3} \rceil\}$ is an opaque quorum system for $\mathcal{B}$, whose load is $\frac{1}{n} \lceil \frac{2n+2f}{3} \rceil$. □

The next theorem proves a resilience bound for opaque quorum systems.

Theorem 5.12 Suppose that $\mathcal{B} = \{B \subseteq U : |B| = f\}$. There exists an opaque quorum system for $\mathcal{B}$ iff $n \geq 5f$.

Proof. That $n \geq 5f$ is sufficient is already demonstrated in Example 5.11 above. Now suppose that $\mathcal{Q}$ is an opaque quorum system for $\mathcal{B}$. Fix any $Q_1 \in \mathcal{Q}$ such that $|Q_1| \leq n - f$ (Q_1 exists by O-Availability); note that $|Q_1| > f$ by O-Consistency2. Choose $B_1 \subseteq Q_1$, $|B_1| = f$, and some $Q_2 \in \mathcal{Q}$ such that $Q_2 \subseteq U \setminus B_1$ (Q_2 exists by O-Availability). Then $|Q_1 \cap Q_2| \leq n - 2f$. By O-Consistency2, $|Q_1 \cap Q_2| \geq f$, and therefore there is some $B_2 \in \mathcal{B}$ such that $B_2 \subseteq Q_1 \cap Q_2$. Then

$$
\begin{aligned}
n - 3f &\geq |Q_2 \cap Q_1| - |B_2| \\
&= |(Q_2 \cap Q_1) \setminus B_2| \\
&\geq |(Q_1 \setminus Q_2) \cup (Q_1 \cap B_2)| \qquad (1) \\
&= |Q_1 \setminus Q_2| + |B_2| \\
&\geq |B_1| + |B_2| \\
&= 2f
\end{aligned}
$$

Where (1) holds by O-Consistency1. Therefore, we have $n \geq 5f$. □

Example 5.13 (*Partition*) Suppose that $\mathcal{B} = \{B_1, \ldots, B_{3k}\}$, $k > 1$, is a partition of U where $B_i \neq \emptyset$ for all i, $1 \leq i \leq 3k$. Choose any collection of sets $\hat{B}_i \subseteq B_i$, $1 \leq i \leq 3k$, such that $|\hat{B}_i| = c$ for a fixed constant $c > 0$. Then, the f-opaque construction of Example 5.11 on the 'super-elements' $\{\hat{B}_i\}$ (as in Example 4.8), with universe size $3k$ and a threshold assumption $f = 1$, yields an opaque quorum system with load $\frac{2k+1}{3k}$. □

Unlike the case for regular masking quorum systems, an open problem is to find a technique for testing whether, given a fail-prone system $\mathcal{B}$, there exists an opaque quorum system for $\mathcal{B}$ (other than an exhaustive search of all subsets of 2^U).

In the constructions in Examples 5.11 and 5.13, the resulting quorum systems exhibited loads that at best were constant as a function of n. In the case of masking quorum systems, we were able to exhibit quorum systems whose load decreased as a function of n, namely the grid quorums. A natural question is whether there exists an opaque quorum system for any fail-prone system $\mathcal{B}$ that has load that decreases as a function of n. In this section, we answer this question in the negative: we show a lower bound of $\frac{1}{2}$ on the load for any opaque quorum system construction, regardless of the fail-prone system.

Theorem 5.14 The load of any opaque quorum system is at least $\frac{1}{2}$.

Proof. O-Consistency1 implies that for any $Q_1, Q_2 \in \mathcal{Q}$, $|Q_1 \cap Q_2| \geq |Q_1 \setminus Q_2|$, and thus $|Q_1 \cap Q_2| \geq \frac{|Q_1|}{2}$. Let w be any strategy for the quorum system $\mathcal{Q}$, and fix any $Q_1 \in \mathcal{Q}$. Then, the total load induced by w on the elements of Q_1 is:

$$\begin{aligned}
\sum_{u \in Q_1} l_w(u) &= \sum_{u \in Q_1} \sum_{Q_i \ni u} w(Q_i) \\
&= \sum_{Q_i} \sum_{u \in Q_1 \cap Q_i} w(Q_i) \\
&\geq \sum_{Q_i} \frac{|Q_1|}{2} w(Q_i) \\
&= \frac{|Q_1|}{2}
\end{aligned}$$

Therefore, there must be some server in Q_1 that suffers a load at least $\frac{1}{2}$. □

We now present a generic construction of an opaque quorum system for $\mathcal{B} = \{\emptyset\}$ and increasingly large universe sizes n, that has a load that tends to $\frac{1}{2}$ as n grows. We give this construction primarily to show that in at least some cases the lower bound of $\frac{1}{2}$ is tight; due to the requirement that $\mathcal{B} = \{\emptyset\}$, this construction is not of practical use for coping with Byzantine failures.

Example 5.15 Suppose that the universe of servers is $U = \{u_1, \ldots, u_n\}$ where $n = 2^\ell$ for some $\ell > 2$, and that $\mathcal{B} = \{\emptyset\}$. Consider the $n \times n$ Hadamard matrix $H(\ell)$, constructed recursively as follows:

$$\begin{aligned}
H(1) &= \begin{bmatrix} -1 & -1 \\ -1 & 1 \end{bmatrix} \\
H(k) &= \begin{bmatrix} H(k-1) & H(k-1) \\ H(k-1) & -H(k-1) \end{bmatrix}, k \geq 2
\end{aligned}$$

$H(\ell)$ has the property that $H(\ell)H(\ell)^T = nI$, where I is the $n \times n$ identity matrix. Using well-known inductive arguments [16, Ch. 14], it can be shown that (i) the first row and column consist entirely of -1's, (ii) the i-th row and i-th column, for each $i \geq 2$, has 1's in $\frac{n}{2}$ positions (and similarly for -1's), and (iii) any two rows (and any two columns) $i, j \geq 2$ have identical elements in $\frac{n}{2}$ positions, i.e., 1's in $\frac{n}{4}$ common positions and -1's in $\frac{n}{4}$ common positions.

We treat the rows of $H(\ell)$ as indicators of subsets of U. That is, let $Q_i = \{u_j : H(\ell)[i,j] = 1\}$ be the set defined by the i-th row, $1 \le i \le n$. Note that $Q_1 = \emptyset$ and that u_1 is not included in any Q_i. We claim that the system $\mathcal{Q} = \{Q_2, ..., Q_n\}$ is an opaque quorum system for $\mathcal{B}$. Using properties (i)–(iii) above, we have that $|Q_i| = \frac{n}{2}$ for each $i \ge 2$; that each u_i, $i \ge 2$, is in exactly $\frac{n}{2}$ of the sets $Q_2, \ldots, Q_n$; and that for any $i, j \ge 2$, if $i \ne j$ then $|Q_i \cap Q_j| = \frac{n}{4}$. From these, the O-Consistency1 and O-Consistency2 requirements can be quickly verified, and a load of $\frac{\frac{n}{2}}{n-1}$ can be achieved, e.g., with a strategy that assigns equal probability to each quorum. □

6 Faulty clients

So far, we have been concerned with providing a consistent service to a set of correct clients. In this section, we extend our treatment to address faulty clients in addition to faulty servers. Since updates may now be generated by faulty clients, we can make no assumption of self-verifying data, and thus use masking quorum systems (Section 4) to implement the service. We focus on ensuring the consistency of the data stored at the replicated service as seen by correct clients only.

A difficulty in handling faulty clients is that a faulty writer might send different updates to different servers and may fail to contact a full quorum. We therefore modify the write protocol to prevent clients from leaving the service in an inconsistent state, and to guarantee that updates propagate to (at least) a full quorum. We maintain availability of the service despite the possibly malicious behavior by any number of clients, so that a correct client can always complete a write operation with as little as one available quorum.

The treatment here provides a single-writer multi-reader safe variable semantics (ignoring reads by faulty clients). Since the initial conference publication of this work [28], single-writer objects with stronger semantics in the case of faulty clients have been constructed using Byzantine quorums and have been used to solve the distributed consensus problem [29]. Other work has extended the treatment here to provide multi-writer variables [30], using a protocol that employs digital signatures and avoids any communication among the servers themselves.

The write protocol performed by a client is changed in that a writer computes the timestamp locally, without consulting the servers, and in that it denotes the quorum it attempts to access in the update request. We replace the write operation of Section 3 by the following:

Write: For a client c to write the value v, it chooses a timestamp $t \in T_c$ greater than any value it has chosen before, and then performs the following two steps: (i) it chooses a quorum Q and sends an update message `<update,`Q, v, t`>` to each server in Q, and (ii) if after some timeout period, it has not received an acknowledgement from every server in Q, it repeats (i) (and (ii)).

Every server that receives an `update` message from a client engages in an "update" protocol to guarantee uniqueness of the value associated with a timestamp and its propagation to a full quorum. The protocol is presented in Figure 5.

In order to argue correctness for this protocol, we have to adapt the definition of operation precedence and operation duration to allow for the behavior of a faulty client. The reason is that it is unclear how to define when an operation by a faulty client begins or ends, as the client can behave outside the specification of any protocol. We make use of the following terminology:

Definition 6.1 We say that a server *delivers* an update $<v, t>$ when it receives <ready, Q, v, t> from each server in the set $Q^- = Q \setminus B$ for some fail-prone set B (step 4 of the update protocol in Figure 5). □

1. If a server receives <update, Q, v, t> from a client c, if $t \in T_c$, and if the server has not previously received from c a message <update, Q', v', t'> where either $t' = t$ and $v' \neq v$ or $t' > t$, then the server sends <echo, Q, v, t> to each member of Q.

2. If a server receives identical echo messages <echo, Q, v, t> from every server in Q, then it sends <ready, Q, v, t> to each member of Q.

3. If a server receives identical ready messages <ready, Q, v, t> from a set B^+ of servers, such that $B^+ \not\subseteq B$ for all $B \in \mathcal{B}$, then it sends <ready, Q, v, t> to every member of Q if it has not done so already.

4. If a server receives identical ready messages <ready, Q, v, t> from a set Q^- of servers, such that for some $B \in \mathcal{B}$, $Q^- = Q \setminus B$, then (i) if t is greater than the timestamp it currently holds, then it updates its variable and timestamp to v and t, respectively, and (ii) regardless of whether it updates the variable and timestamp, it sends an acknowledgment message to c where $T_c \ni t$.

Figure 5: An update protocol

We now say that a write operation that writes v with timestamp t begins when the first correct server receives <update, Q, v, t>, and ends when all correct servers in some quorum have delivered the update. Note that by this definition, a write operation by a faulty client could last arbitrarily long, and could overlap other writes by the same client. Nevertheless, carrying over the remainder of the precedence definition, we have that the write protocol together with the update protocol in Figure 5 implement a single-writer multi-reader safe variable:

Lemma 6.2 A correct process' read operation that is concurrent with no write operations returns the value written by the last preceding write operation in some serialization of all preceding write operations.

To prove this lemma, we need the following properties of our protocol:

Lemma 6.3 A correct server delivers <v, t> only if some correct server previously received <update, Q, v, t>.

Proof. To deliver <v, t>, a correct server must receive a ready message from some correct server. Moreover, the first <ready, Q, v, t> message from a correct server is sent only after it receives <echo, Q, v, t> from each member of Q. Since, a correct member sends <echo, Q, v, t> only if it first receives <update, Q, v, t>, this proves the lemma. □

Lemma 6.4(*Agreement*) If a correct server delivers <v, t> and a correct server delivers <v', t>, then $v = v'$.

Proof. As argued in the previous lemma, for a correct server to deliver <v, t>, <echo, Q, v, t> must have been sent by all servers in Q. Similarly, <echo, Q', v', t> must have been sent by all servers in Q'. Since every two quorums intersect in (at least) one correct server, and since any correct server sends <echo, $*$, $\hat{v}$, t> for at most one value $\hat{v}$, v must be identical to v'. □

Proof of Lemma 6.2. Let W denote the set of write operations preceding the read. Note that by Lemma 6.4, any value/timestamp pair in W is well defined, i.e., the same value corresponds to any timestamp at all correct servers that deliver it. By definition, every write in W was delivered to a full quorum, and by assumption and Lemma 6.3, no correct server has delivered any write outside W. Therefore, by the construction of masking quorum systems, the read operation will return the value written in the write operation in W with the highest timestamp. So, it suffices to argue that there is a serialization of the writes in W in which this write operation appears last, or in other words, that this write operation precedes no other write operation in W. This results, however, from the fact that there is a single writer and that servers echo an update request only if its timestamp is higher than the one they have in store, and so any later write operation has a higher timestamp. $\square$

In addition, we argue liveness and completeness of our protocol as follows:

Lemma 6.5(*Propagation*) If a correct server delivers <v, t>, then eventually there exists a quorum $Q \in \mathcal{Q}$ such that every correct server in Q delivers <v, t>.

To prove this lemma, we make use of the following fact:

Lemma 6.6 If $\mathcal{Q}$ is a masking quorum system over a universe U with respect to a fail-prone system $\mathcal{B}$, then $\forall Q \in \mathcal{Q}\ \forall B_1, B_2, B_3 \in \mathcal{B}$, $Q \not\subseteq B_1 \cup B_2 \cup B_3$.

Proof. Assume otherwise for a contradiction, i.e., that there is a $Q \in \mathcal{Q}$ and $B_1, B_2, B_3 \in \mathcal{B}$ such that $Q \subseteq B_1 \cup B_2 \cup B_3$. By M-Availability, there exists $Q' \in \mathcal{Q}$, $Q' \cap B_1 = \emptyset$. Then, $Q \cap Q' \subseteq B_2 \cup B_3$ and thus $(Q \cap Q') \setminus B_2 \subseteq B_3$, contradicting M-Consistency. $\square$

Proof of Lemma 6.5. According to the protocol, the correct server that delivered <v, t> received a message <ready, Q, v, t> from each server in $Q^- = Q \setminus B$ for some $Q \in \mathcal{Q}$ and $B \in \mathcal{B}$. Since, for some $B' \in \mathcal{B}$, (at least) all the members in $Q^- \setminus B'$ are correct, every correct member of Q receives <ready, Q, v, t> from each of the members of $B^+ = Q^- \setminus B'$. Since, $\forall B'' \in \mathcal{B}$, $Q^- \setminus B' \not\subseteq B''$ (by Lemma 6.6), the ready messages from B^+ cause each correct member of Q to send such a ready message. Consequently, <v, t> is delivered by all of the correct members of Q. $\square$

Lemma 6.7(*Validity*) If a correct client c sends <update, Q, v, t> to every server in Q and all servers in Q are correct, then eventually a correct server delivers <v, t>.

Proof. Since both the client and all of the members of Q are correct, <update, Q, v, t> will be received and echoed by every member in Q. Consequently, all the servers in Q will send <ready, Q, v, t> messages to the members of Q, and will eventually deliver <v, t>. $\square$

7 Conclusions

The literature contains an abundance of protocols that use quorums for accessing replicated data. This approach is appealing for constructing replicated services as it allows for increasing the availability and efficiency of the service while maintaining its consistency. Our work extends this successful approach to environments where both the servers and the clients of a service may deviate from their prescribed behavior in arbitrary ways. We introduced a new class of quorum systems, namely *masking* quorum systems, and devised protocols that use these quorums to enhance the availability of systems prone to Byzantine failures. We also explored two variations of our quorum systems, namely *dissemination* and *opaque masking* quorums, and for all of these classes of quorums we provided various constructions and analyzed the load they impose on the system.

Our work leaves a number of intriguing open challenges and directions for future work. One is to characterize the average performance of our quorum constructions and their load in less-than-ideal scenarios, e.g., when failures occur. Also, in this work we described only quorum systems that are uniform, in the sense that any quorum is possible for both read and write operations. In practice it may be beneficial to employ quorum systems with distinguished *read quorums* and *write quorums*, with consistency requirements imposed only between pairs consisting of at least one write quorum. Although this does not seem to improve our lower bounds on the overall load that can be achieved, it may allow greater flexibility in trading between the availability of reads and writes.

Acknowledgments

We are grateful to Andrew Odlyzko for suggesting the use of Hadamard matrices to construct opaque masking quorum systems with an asymptotic load of $\frac{1}{2}$. We also thank Yehuda Afek and Michael Merritt for helpful discussions, and Vassos Hadzilacos and Rebecca Wright for many helpful comments on earlier versions of this paper. An insightful comment by Rida Bazzi led to a substantial improvement over a previous version of this paper.

References

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt and N. Shavit. Atomic snapshots of shared memory. *Journal of the ACM* 40(4):873–890, September 1993.

[2] Y. Afek, D. Dolev, E. Gafni, M. Merritt and N. Shavit. A bounded first-in first-enabled-solution to the l-exclusion problem. In *Proceedings of the 4th International Workshop on Distributed Algorithms*, LNCS 486, Springer-Verlag, 1990.

[3] D. Agrawal and A. El Abbadi. Integrating security with fault-tolerant distributed databases. *Computer Journal* 33(1):71–78, February 1990.

[4] D. Agrawal and A. El Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM Transactions on Computer Systems* 9(1):1–20, 1991.

[5] J. H. Anderson. Composite registers. *Distributed Computing* 6(3):141–154, 1993.

[6] R. A. Bazzi. Synchronous Byzantine quorum systems. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 259–266, August 1997.

[7] P. A. Bernstein, V. Hadzilacos and N. Goodman. *Concurrency control and recovery in database systems.* Addison-wesley, 1987.

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

[9] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. In *Proceedings of the 6th IEEE International Conference on Data Engineering*, pages 438–445, 1990.

[10] International Telegraph and Telephone Consultative Committee (CCITT). The Directory – Authentication Framework, Recommendation X.509, 1988.

[11] D. Dolev, E. Gafni and N. Shavit. Toward a non-atomic era: l-exclusion as a test case. In *Proceedings of the 20th ACM Symposium on Theory of Computing*, pages 78–92, May 1988.

[12] D. Dolev and N. Shavit. Bounded concurrent time-stamp systems are constructible. *SIAM Journal of Computing*, to appear. Also in *Proceedings of the 21st ACM Symposium on the Theory of Computing*, pages 454–466, 1989.

[13] A. El Abbadi and S. Toueg. Maintaining availability in partitioned replicated databases. *ACM Transactions on Database Systems* 14(2):264–290, June 1989.

[14] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM* 32(4):841–860, October 1985.

[15] D. K. Gifford. Weighted voting for replicated data. In *Proceedings of the 7th ACM Symposium on Operating Systems Principles*, pages 150–162, 1979.

[16] M. Hall, Jr. *Combinatorial Theory.* 2nd Ed. Wiley-Interscience Series in Discrete Mathematics, 1986.

[17] M. Herlihy. A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems* 4(1):32–53, February 1986.

[18] M. P. Herlihy and J. D. Tygar. How to make replicated data secure. In *Advances in Cryptology—CRYPTO '87 Proceedings* (Lecture Notes in Computer Science 293), pages 379–391, Springer-Verlag, 1988.

[19] A. Israeli and M. Li. Bounded time-stamps. *Distributed Computing* 6(4):205–209.

[20] A. Kumar. Hierarchical quorum consensus: A new algorithm for managing replicated data. *IEEE Transactions on Computers* 40(9):996–1004, 1991.

[21] A. Israeli and A. Shaham. Optimal multi-write multi-reader atomic register. In *Proceedings of the 11th ACM Symposium on Principles of Distributed Computing*, pages 71–82, 1992.

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

[23] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems* 10(4):265–310, November 1992.

[24] L. Lamport. On interprocess communication (part II: algorithms). *Distributed Computing* 1:86–101, 1986.

[25] M. Li, J. Tromp and P. M. B. Vitanyi. How to share concurrent wait-free variables. *Journal of the ACM*, to appear.

[26] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems* 3(2):145–159, 1985.

[27] D. Malkhi and M. Reiter. A high-throughput secure reliable multicast protocol. *Journal of Computer Security*, 5, 1997, pp 113-127.

[28] D. Malkhi and M. Reiter. Byzantine quorum systems. In *Proceedings of the 29th ACM Symposium on Theory of Computing (STOC)*, pages 569–578, May 1997.

[29] D. Malkhi and M. Reiter. Survivable consensus objects. In preparation.

[30] D. Malkhi and M. Reiter. Secure and scalable replication in Phalanx. In preparation.

[31] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 249–257, August 1997.

[32] D. Malkhi, M. Reiter, and R. Wright. Probabilistic quorum systems. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 267–273, August 1997.

[33] D. Malkhi, M. Reiter, A. Wool and R. Wright. Probabilistic Byzantine Quorum Systems. **Brief Announcement** in *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing (PODC)*, 1998, to be published. Full version submitted for publication.

[34] R. Mukkamala. Storage efficient and secure replicated distributed databases. *IEEE Transactions on Knowledge and Data Engineering* 6(2):337–341, April 1994.

[35] 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.

[36] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. In *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pages 157–168, March 1996.

[37] M. O. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM* 36(2):335–348, 1989.

[38] M. K. Reiter. Distributing trust with the Rampart toolkit. *Communications of the ACM* 39(4):71–74, April 1996.

[39] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM* 21(2):120–126, February 1978.

[40] A. Shamir. How to share a secret. *Communications of the ACM* 22(11):612–613, November 1979.

[41] J. J. Tardo and K. Alagappan. SPX: Global authentication using public key certificates. In *Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy*, pages 232–244, May 1991.

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