Synchronous Byzantine Quorum Systems

Rida A. Bazzi
Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406 U.S.A.
bazzi@asu.edu

Abstract

Quorum systems have been used to implement many coordination problems in distributed systems such as mutual exclusion, data replication, distributed consensus, and commit protocols. Malkhi and Reiter recently proposed quorum systems that can tolerate byzantine failures; they called these systems byzantine quorum systems and gave some examples of such quorum systems. In this paper, we argue that the proposed definition of byzantine quorums is too strong for synchronous systems and we propose a definition of synchronous byzantine quorums. We show that the new definition is more appropriate for synchronous systems. We prove tight lower bounds on the load of synchronous byzantine quorums for various failure assumptions and we present synchronous byzantine quorums that have optimal loads that match the lower bounds for two failure assumptions.

Keywords: quorum systems, fault tolerance, byzantine failures, load, distributed systems.

1 Introduction

A quorum system is a collection of sets (quorums) that mutually intersect. Quorum systems have been used to implement mutual exclusion [1, 9], replicated data systems [8], commit protocols [19], and distributed consensus [13]. For example, in a typical implementation of mutual exclusion using a quorum system, processors request access to the critical section from all members of a quorum. A processor can enter its critical section only if it receives permission from all processors in a quorum.[1] Quorum systems are usually evaluated according to the following criteria:

1. Quorum Size. The size of a quorum determines the number of messages needed to access it.

2. Load. Informally, the load of a processor measures the share it has in in handling requests to the quorum system. For a given probabilistic access rule of a quorum system, the load of a quorum system is the probability of accessing the busiest processor. The load of a quorum system is the minimum load over all access probability distributions.

3. Availability and Failure Probability. A quorum system is said to be available if one of its quorums consists of correct processors. If failures occur randomly according to some probability distribution, the failure probability is the probability that the quorum is not available. The probabilistic model is not appropriate for byzantine failures.

Other criteria are also considered in the literature, but we do not study them in this paper.

Work on quorum systems traditionally considered crash failures [1, 2, 3, 4, 7, 8, 9, 18, 17, 16]. Recently, Malkhi and Reiter [10] proposed the interesting notion of byzantine quorums - quorum systems that can tolerate byzantine failures. They showed that the traditional definition of quorums is not adequate for byzantine quorums: in byzantine quorum systems, the intersection of two quorums should contain enough correct processors so that correct processors can unambiguously access the quorums. They

[1] Additional measures are needed to insure that the implementation is fair and deadlock free.

called these quorums masking quorums because they hide the failures from processors using them. They also proposed variations of the basic byzantine quorum system for the case in which responses by faulty processors can be authenticated; they called these systems dissemination quorum systems. Finally, they proposed the notion of opaque byzantine quorums and proved lower and upper bounds on their load. They presented a number of masking byzantine and dissemination quorum systems for different failure assumptions and calculated their load. The quorum systems they presented assume that the system is asynchronous.

2 Contributions

In this paper we carefully study the definitions of byzantine quorums as proposed in [10]. Specifically, we study how the synchrony assumptions affect the definition of byzantine quorums. This leads us to revisit the traditional definition of availability of quorum systems and the way quorums are accessed. We find that the traditional definition of availability is intended to guarantee the correctness of quorum-based protocols in a model in which failed processors can recover and, therefore, allows for a probabilistic model of failures. We note that the probabilistic model is not adequate for byzantine failures and conclude that a new definition of availability and quorum access is needed in the presence of byzantine failures in a synchronous system. We revise the definition of masking byzantine quorums proposed in [10] and propose to relax it in a way that allows for more failures in synchronous systems. We develop necessary and sufficient conditions for the existence of synchronous byzantine quorums that can tolerate a general failure pattern. We develop tight lower bounds for the load of synchronous byzantine quorums for various failure patterns and, for each lower bound, we present a quorum system whose load is optimal and matches the lower bound.

The rest of the paper is organized as follows. Section 3 presents some basic definitions and introduces the notions of quorum access and availability. Section 4 argues that, in synchronous systems, the definition of byzantine quorum systems should be relaxed and presents a relaxed definition. Section 5 proves lower bounds on the load of synchronous byzantine quorum systems for various failure assumptions. Section 6 presents synchronous byzantine quorum systems that have optimal loads. Section 7 presents a general construction of byzantine quorum systems.

3 Definitions and System Model

3.1 System Model

We assume that the system consists of a set $\mathcal{P}$ of n server processors and a number of client processors that are distinct from the servers. All processors can communicate using reliable message passing. We also assume that there is a bound on message delivery time that can be used to timeout failed processors that do not respond to requests.

3.2 Failure Model

Server processors can fail.[2] The assumptions about failures affect the way a quorum can be used. In studying quorum systems, it is usually assumed that processors fail randomly according to some probability distribution and that failures are temporary. Assuming that processors fail randomly is appropriate for crash failures or for failures that can be detected. A different model of failures is usually used for byzantine (arbitrary) failures. In the byzantine failure model, it is usually assumed that there is a *known bound* on the number of failures that can occur in the system and that failed processors do not recover. This is the model we adopt in this paper. The set *faulty* denotes the set of faulty processors in the system. A *failure pattern* $\mathcal{F}$ identifies the possible sets of faulty processors in the system. We write $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$. There exists an element F of $\mathcal{F}$ such that at any given instant the faulty processors belong to F. The processors do not necessarily know F. A common example of a failure pattern is the f-*threshold* pattern in which $\mathcal{F} = \{F \in \mathcal{P} : \ |F| = f\}$. Another interesting failure pattern is the *disjoint* pattern in which all elements of $\mathcal{F}$ are disjoint [10].

3.3 Coteries and Quorums

The standard definition of a quorum is the following.

Definition 1: *A* quorum system $\mathcal{Q}$ *over* $\mathcal{P}$ *is a set of subsets (called* quorums*) of* $\mathcal{P}$ *such that any two quorums have a non-empty intersection.*

The intersection property of quorums is essential for their use in coordination problems.

Definition 2: *A* coterie $\mathcal{C}$ *over* $\mathcal{P}$ *is a quorum system over* S *which is minimal under set inclusion.*

[2] We do not consider client failures in this paper.

In this paper, we only consider quorum systems that are coteries.

Definition 3: *A quorum system $\mathcal{Q}$* dominates *a quorum system $\mathcal{Q}'$ if $\mathcal{Q} \neq \mathcal{Q}'$ and for every $Q' \in \mathcal{Q}'$ there exists $Q \in \mathcal{Q}$ such that $Q \subseteq Q'$.*

This suggests the definition of a *non–dominated quorum systems*, or *NDQ* for short.

Definition 4: *A quorum system is* non–dominated *if there is no other quorum system that dominates it.*

3.4 Strategies and Load

This section presents the formal definitions of strategy and load [15].

A protocol using a quorum system chooses a quorum for access according to some rules. A strategy is a probabilistic rule to choose a quorum. Formally, a strategy is defined as follows.

Definition 5: *Let $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ be a quorum system. A* strategy *w for $\mathcal{Q}$ is a probability distribution over $\mathcal{Q}$.*

The probability of accessing Q_i is $w(Q_i)$. For every processor $q \in \mathcal{P}$, a strategy w induces a probability that q is accessed. This probability is called the load on q. The *system load* is the load on the *busiest* element induced by the best possible strategy.

Definition 6: *Let w be a strategy for a quorum system $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$. For any $q \in \mathcal{P}$, the* load *induced by w on q is $l_w(q) = \sum_{Q_j \ni q} w(Q_j)$. The* load *induced by w on $\mathcal{Q}$ is*

$$\mathcal{L}_w(\mathcal{Q}) = \max_{q \in \mathcal{P}} l_w(q)$$

The system load *on $\mathcal{Q}$ is*

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

where the minimum is taken over all strategies.

3.5 Quorum Access and Availability

Processors access a quorum to coordinate their actions. The following is the typical way to access a quorum. A processor, the *client*, sends a request to every processor, *server*, in a quorum set. Upon receiving a request, a correct processor updates its state and sends a reply. The client waits until it receives a reply from every processor in the quorum. The replies typically reflect requests made by other processors. If the client receives replies from all processors in a quorum then the access is considered successful. If one of the servers failed, then the client attempts to access another quorum that does not have any faulty processor (the question of finding a quorum with no faulty processors has been addressed in [2, 16]). Since processors access a quorum only if all its members are correct, two clients are always guaranteed to receive a response from a common correct server which belongs to a non-empty intersection of two quorums. The correctness of quorum-based protocols rely on this intersection property. Traditionally, a quorum is said to be *available* if all its elements are correct processors [15]. One might think that it is sufficient to have a non-empty intersection between quorums to have availability. The following is an alternative definition of availability.

- **Availability (alternative)**. A quorum system is available if one of its quorums intersects every other quorum in a correct processor.

This definition seems to guarantee the non-empty intersection property of quorum systems. Unfortunately, this definition is not sufficient if processors can recover from failures. The following example clarifies this point. Consider the quorum system $\mathcal{Q} = \{\{1,2,3,4\}, \{1,2,5,6\}, \{2,3,5,6\}\}$. This system is such that any two quorums have two processors in common. Assume that initially processors 1, 3, and 5 are faulty. Trying to get permission to enter its critical section, a processor accessing quorum $\{1,2,3,4\}$ would get permission from $\{2,4\}$, a set that intersects every other quorum in a correct processor. Assume that after this request is granted processors 2, 4, and 6 become faulty and that processors 1, 3, and 5 recover. Now, another processor trying to access its critical section might be granted permission by the set $\{1,3\}$, a set that intersects every other quorum. If the alternative definition of availability is adopted, we will have two processors in the critical section. Nevertheless, the alternative definition can be used if failures are not temporary. To access a quorum in a synchronous system, a processor sends a request and waits to receive replies or timeouts for all elements of a quorum set[3]. If the set of processors that replied intersects every other quorum, then the replies can be used to coordinate. This is because processors that have failed will not recover later and put the system in an inconsistent state. This is the case if failures are arbitrary.

[3] Timeouts are not possible in asynchronous systems

4 Byzantine Quorums

Malkhi and Reiter [10] gave a general definition of quorum systems that can tolerate byzantine failures in asynchronous systems. We will first discuss their definition, then we propose a less restrictive definition that works in synchronous systems.

4.1 Byzantine Quorums in Asynchronous Systems

If failures are arbitrary, a processor might receive conflicting replies from faulty and correct processors. It follows that a processor must base its coordination decisions on replies that it *knows* to be from correct processors. Motivated by this requirement, Malkhi and Reiter gave the following definition [10]:

Definition 7: *A quorum system tolerates failure pattern $\mathcal{F}$ if*

1. $\forall Q_1, Q_2 \in \mathcal{Q} \;\; \forall F_1, F_2 \in \mathcal{F} : (Q_1 \cap Q_2) - F_1 \not\subseteq F_2$.
2. $\forall F \in \mathcal{F} \;\; \exists Q \in \mathcal{Q} : F \cap Q = \emptyset$

The first condition requires that the intersection of two quorums is not contained in the intersection of two sets in $\mathcal{F}$. This will guarantee that the replies of correct processors can be identified . To see why this is the case, consider, as an example, the f-threshold failure pattern. For that pattern, the condition reduces to a familiar-looking condition: $Q_1 \cap Q_2 \geq 2f+1$. So, it is always possible to have $f+1$ identical replies in the intersection of two quorums. One of these replies must be the reply of a correct processor. It should not be surprising that, even though the failures are arbitrary, only a simple majority of correct processors (as opposed to a two-third majority) is required in the intersection. This is due to the fact that the clients are assumed to be correct. The second condition is based on the traditional definition of availability; it requires that some quorum consists of correct processors. As we mentioned above, this definition of availability is designed for a failure model in which faulty processor can recover, which is not the case for arbitrary failures. Nevertheless, in asynchronous systems, the second condition is needed because there is no way to differentiate a slow processor from a faulty one. To access a quorum in an asynchronous system, a processor cannot simply send requests to all processors in a quorum set and waits for replies. In the worst case, in a failure-free execution, a processor might have to send requests to every processor in the system and then waits for replies from a quorum that consists of correct processor. The availability condition is needed to guarantee that some response is received. Sending requests to every processor limits the efficiency of the use of quorums in asynchronous systems. Sending requests to every processor is not needed in synchronous systems if no failures occur. In what follows, we will refer to byzantine quorums for asynchronous systems as asynchronous byzantine quorum systems.

4.2 Byzantine Quorums in Synchronous Systems

We should first point to an important distinction between quorum systems that tolerate byzantine failures and those that tolerate crash failures. If failures are benign, then the availability property can be checked *a posteriori*: a client can determine if a quorum system is available or not. This ability of determining the availability of the quorum system is not possible if failures are byzantine. If many failures occur and the system is not available, then a client cannot determine that fact because, in general, there is no way to distinguish a correct processor from a faulty one. It follows that the availability condition of a byzantine quorum should hold *a priori*. A byzantine quorum system designed for a given failure pattern cannot dynamically adapt if the failures are different from those specified by the pattern it is designed to tolerate.

From the discussion above and in the previous section, we conclude that only the first of the two conditions proposed by [10] is needed for byzantine quorums in synchronous systems.

Definition 8: *For byzantine failures, a quorum system tolerates failure pattern $\mathcal{F}$ if:*

$$\forall Q_1, Q_2 \in \mathcal{Q} \;\; \forall F_1, F_2 \in \mathcal{F} : (Q_1 \cap Q_2) - F_1 \not\subseteq F_2$$

Note that this definition, which is identical to the first condition of Definition 7, requires that the failure pattern be known *a priori*. It is enough to guarantee that the quorum system can handle the worst-case failure scenario. That's why we do not have to require an extra condition for availability. Note that a synchronous byzantine quorum system is not necessarily an asynchronous byzantine quorums system. On the other hand, an asynchronous byzantine quorum system is also a synchronous byzantine quorum system.

To see how a quorum satisfying Definition 8 can be used to coordinate in the presence of byzantine failures, we consider the implementation of a sequentially consistent replicated single bit register that can be set to 1 only once and whose initial value is 0. To set the register, a client sends a *write* message to every processor in a quorum. On receipt of the *write* message, a server will change its local copy of the register to 1. To read the value of the register, a client sends a message requesting the value from every processor in a quorum. If the client receives identical replies from a set of processors that is not a subset of a faulty set, then it *reads* the value in the identical replies, otherwise it *reads* 0. For a sketch of the proof of correctness, we note that if a write precedes a read, then the read will return 1 because the intersection of two quorums cannot be covered by two faulty set. Also, there are not enough faulty processors in the intersection of two quorums to make a read return 1 if it is not preceded by a write. Note that the implementation still works even if every quorum has some faulty processors (system is not available according to the traditional definition of availability). This is due to the fact that Definition 8 *a priori* handles the worst failure scenario assumed in the system.

Given a failure pattern, we are interested in deciding whether there exists a quorum system that tolerates the failure pattern. The following two propositions give necessary and sufficient conditions.

Proposition 9: *There exists a quorum system that tolerates failure pattern $\mathcal{F}$ if and only if $\mathcal{Q} = \{\mathcal{P}\}$ tolerates $\mathcal{F}$.*

Proof: If $\mathcal{Q} = \{\mathcal{P}\}$ tolerates $\mathcal{F}$, then there exists a quorum system that tolerates $\mathcal{F}$. If there exists a quorum system $\mathcal{Q}$ that tolerates $\mathcal{F}$, then there exists a quorum in $\mathcal{Q}$ that cannot be contained in the union of any two elements of $\mathcal{F}$. It follows that $\mathcal{P}$ cannot is not equal to the union of any two elements in $\mathcal{F}$ and that $\{\mathcal{P}\}$ tolerates $\mathcal{F}$. ∎

Proposition 10: *There exists a quorum system that tolerate failure pattern $\mathcal{F}$ if and only if $\forall F_1, F_2 \in \mathcal{F}$: $\mathcal{P} \neq F_1 \cup F_2$.*

Proof: Direct application of Proposition 9. ∎

It follows that there exists a quorum system that tolerates the f-threshold failure pattern if and only if $n \geq 2f + 1$. This necessary and sufficient condition is to be contrasted with the requirement that $n \geq 4f + 1$ for the existence of a byzantine quorum for asynchronous systems [10].

5 Load Lower Bounds

Let $c(\mathcal{Q})$ be the size of the smallest quorum of $\mathcal{Q}$. Naor and Wool[15] proved the following lower bound on the load of a quorum system.

Proposition 11: $\mathcal{L}(\mathcal{Q}) \geq \max\{\frac{c(\mathcal{Q})}{n}, \frac{1}{c(\mathcal{Q})}\}$.

This lower bound obviously applies to byzantine quorum systems because they are also quorum systems. Malkhi and Reiter [10] proved the same lower bounds with more direct methods, but did not prove any new lower bounds for byzantine quorums.[4] In this section, we prove new lower bounds that are specific to byzantine quorums in synchronous systems. The lower bounds we prove also apply to asynchronous byzantine quorums [10]. We prove lower bounds for a general failure pattern, the f-threshold failure pattern, and the disjoint failure pattern. The bounds we prove for the f-threshold failure pattern and the disjoint failure pattern are tight. In Section 6, we present quorum constructions whose loads match the bounds for the f-threshold failure pattern and the disjoint failure pattern.

5.1 General Case

We first present a new form for Definition 8.

Proposition 12: *Let $\mathcal{Q}$ be a quorum system that tolerates failure pattern $\mathcal{F}$, then*

$$\forall Q_1, Q_2 \in \mathcal{Q} \ \forall F_1, F_2 \in \mathcal{F} :$$

$$(Q_1 \cap (\mathcal{P} - (F_1 \cup F_2)) \cap (Q_2 \cap (\mathcal{P} - (F_1 \cup F_2)) \neq \emptyset$$

Proof: Uses standard manipulations on set operators. ∎

In other words, if $\mathcal{Q}$ tolerates $\mathcal{F}$, then the projection of $\mathcal{Q}$ into $\mathcal{P} - (F_1 \cup F_2)$ is a quorum system on $\mathcal{P} - (F_1 \cup F_2)$, where F_1 and F_2 are elements of $\mathcal{F}$.

Let $\mathcal{Q} - (F_1 \cup F_2) = \{Q - (F_1 \cup F_2) : \ Q \in \mathcal{Q}\}$. be the projection of $\mathcal{Q}$ on $\mathcal{P} - (F_1 \cup F_2)$. The following two propositions follow from the application of Proposition 11 to $\mathcal{Q} - (F_i \cup F_j)$ for any two elements F_i and F_j of $\mathcal{F}$.

Proposition 13: *Let $\mathcal{Q}$ be a quorum system that tolerates failure pattern $\mathcal{F}$, then $\mathcal{L}(\mathcal{Q}) \geq \max\{\frac{c(\mathcal{Q}-(F_i \cup F_j))}{n}, \frac{1}{c(\mathcal{Q}-(F_i \cup F_j))}\}$ for any F_i and F_j.*

Proposition 14: *Let $\mathcal{Q}$ be a quorum system that tolerates failure pattern $\mathcal{F}$, then $\mathcal{L}(\mathcal{Q}) \geq \frac{1}{\sqrt{n - \max_{i,j=1,\ldots,m} |F_i \cup F_j|}}$.*

[4] In [11], Malkhi et al. prove some lower bounds similar to ours.

5.2 Disjoint Failure Pattern

Applying Proposition 13 to a disjoint failure pattern $\mathcal{F}_d$, we obtain:

Proposition 15: *Let $\mathcal{Q}$ be a quorum system that tolerates $\mathcal{F}_d = \{F_1, \ldots, F_m\}$ and assume that the F_i's are sorted in a decreasing order of size, then $\mathcal{L}(\mathcal{Q}) \geq \frac{1}{\sqrt{n-(|F_1|+|F_2|)}}$.*

Proof: Since the sets are disjoint, the cardinality of the union of any two distinct sets is equal to the sum of the cardinalities of the two sets. ■

5.3 Threshold Failure Pattern

In this section, we assume that up to f processors can be faulty. The following lower bound holds for dissemination quorum systems.

Proposition 16: *Let $\mathcal{Q}$ be a byzantine quorum system that tolerates the f-threshold pattern, then $L(\mathcal{Q}) \geq \max\{\frac{c(\mathcal{Q})}{n}, \frac{2f+1}{c(\mathcal{Q})}\}$.*

Proof: By Proposition 13, $L(\mathcal{Q}) \geq \frac{c(\mathcal{Q})}{n}$. So, we only need to prove that $\mathcal{L}(\mathcal{Q}) \geq \frac{2f+1}{c(\mathcal{Q})}$. Let w be any strategy for the quorum system $\mathcal{Q}$. Fix $Q_1 \in \mathcal{Q}$ such that $|Q_1| = c(\mathcal{Q})$. Summing the loads induced by w on all the elements of Q_1, we obtain:

$$\sum_{u \in Q_1} l_w(u) = \sum_{u \in Q_1} \sum_{Q_i \ni u} w_i =$$

$$\sum_{Q_i} \sum_{u \in Q_1 \cap Q_i} w(Q_i) \geq \sum_{Q_i} (2f+1) w_i = 2f+1.$$

It follows that there exists an element in Q_1 that suffers a load of $\frac{f}{c(Q_1)}$. The calculation above can be understood as follows. Each quorum Q_i puts a load of w_i on each element it has in common with Q_1. Since it has $2f+1$ elements in common with Q_1, it follows that it contributes a load of $(2f+1)w_i$ to the sum of the loads of all elements of Q_1. The contributions of all quorums to the sum of the loads of elements in Q_1 is $\Sigma_{Q_i} f w_i = 2f+1$. ■

Note that for $f = 0$, Proposition 16 reduces to Proposition 13. The following lower bound is a corollary of Proposition 16.

Corollary 17: $L(\mathcal{Q}) \geq \sqrt{\frac{2f+1}{n}}$.

Proof: We have two cases to consider. If $c(\mathcal{Q}) \geq \sqrt{n(2f+1)}$, $L(\mathcal{Q}) \geq \frac{c(\mathcal{Q})}{n} \geq \sqrt{\frac{2f+1}{n}}$. If $c(\mathcal{Q}) \leq \sqrt{n(2f+1)}$, $L(\mathcal{Q}) \geq \frac{2f+1}{c(\mathcal{Q})} \geq \frac{2f+1}{\sqrt{n(2f+1)}} = \sqrt{\frac{2f+1}{n}}$. So, in all cases, $L(\mathcal{Q}) \geq \sqrt{\frac{2f+1}{n}}$. ■

6 Load Upper Bounds

In this section, we present synchronous quorum constructions that have loads that match the lower bounds proved in the previous section.

6.1 Threshold Failure Pattern

In this section, we prove that the lower bound developed for the threshold failure pattern is tight.

Proposition 18: *There exists a synchronous quorums system whose load is $\sqrt{\frac{2f+1}{n}}$*

Proof: We construct a system whose load matches the lower bound. Consider the following system, which we call the *parallel finite projective planes*. For simplicity, we assume that n is divisible by $2f+1$ and that $\frac{n}{2f+1} = k^2 + k + 1$ for some $k = p^i$ and some prime number p. It follows that $k \approx \sqrt{\frac{n}{2f+1}}$. Divide the processors into $2f+1$ identical sets. On each of the sets, define a finite projective plane system consisting of $\frac{n}{2f+1}$ quorums numbered from 1 to $k+1$ [9]. Each quorum consists of $k+1$ processors and each processor belongs to exactly $k+1$ quorums [9]. A quorum of the parallel finite projective planes is the union of $2f+1$ identically numbered quorums, one from each set. The size of each quorum is $\approx (2f+1)\sqrt{\frac{n}{2f+1}} = \sqrt{n(2f+1)}$ and each processor belongs to exactly $k+1$ quorums. If we choose quorums with probability $\frac{1}{\frac{n}{2f+1}} = \frac{1}{k^2+k+1}$, we obtain a load of $\frac{k+1}{k^2+k+1} \approx \sqrt{\frac{2f+1}{n}}$ which matches the lower bound for the f-threshold pattern. ■

Corollary 19: *The bound of Corollary 17 is tight.*

Note that the parallel finite projective plane system is also an asynchronous byzantine quorum system. It follows that the bound of Corollary 17 is also tight for asynchronous byzantine quorum systems.

6.2 Disjoint Failure Pattern

In this section, we prove that the lower bound developed for the disjoint failure pattern is tight by constructing a system whose load matches the lower bound of Proposition 15. Let $\mathcal{F} = \{F_1, F_2, \ldots, F_m\}$ be the set of failure sets ordered in decreasing size. Let $\alpha = n - (|F_1| + |F_2|)$. By Proposition 15, the load of a byzantine quorum that tolerates $\mathcal{F}$ is greater than $\frac{1}{\sqrt{\alpha}}$.

Define m_1 and m_2 as follows:

- $m_1 = \min\{j : |F_1 \cup \bigcup_{3\le k\le j+1} F_k| \ge \frac{\alpha}{3}\}$
- $m_2 = \min\{j : |F_2 \cup \bigcup_{m_1 < k\le j+1} F_k| \ge \frac{\alpha}{3}\}$

Note that j and k are bound variables in the definitions of m_1 and m_2. Also, m_1 and m_2 are always guaranteed to exist.

Now, define the three sets S_0, S_1 and S_2 as follows:

- $S_0 = F_1 \cup \bigcup_{3\le k\le m_1} F_k$
- $S_1 = F_2 \cup \bigcup_{m_1 < k\le m_2} F_k$
- $S_2 = \bigcup_{m_2 < k} F_k = \mathcal{P} - (S_1 \cup S_2)$

Note that if $F_1 \ge \frac{\alpha}{3}$, then $S_0 = F_1$. Also, if $F_2 \ge \frac{\alpha}{3}$, then $S_1 = F_2$

Proposition 20: *S_0, S_1 and S_2 are disjoint.*

Proof: Follows immediately from the definition of S_0, S_1 and S_2 and the fact that $\mathcal{F}$ consists of disjoint sets. ∎

Proposition 21: *$S_i \ge \frac{\alpha}{6}$ for $i = 0, 1, 2$.*

Proof: There are three cases to consider depending on the size of F_1 and F_2.

- $F_1 \ge \frac{\alpha}{3}$ and $F_2 \ge \frac{\alpha}{3}$. It follows that $S_0 = F_1$, $S_1 = F_2$, and $S_2 = \mathcal{P} - F_1 \cup F_2$. So, $|S_0| \ge \frac{\alpha}{6}$, $|S_1| \ge \frac{\alpha}{6}$, and $|S_2| = \alpha \ge \frac{\alpha}{6}$.
- $F_1 \ge \frac{\alpha}{3}$ and $F_2 < \frac{\alpha}{3}$. We first note that in this case, $S_0 = F_1$ and $|S_0| = |F_1|$. We know that $|S_1 \cup F_{m_2+1}| \ge \frac{\alpha}{3}$ and $|S_1| < \frac{\alpha}{3}$. Also, $|S_1| \ge |F_{m_2+1}|$ because the F_i's are sorted in decreasing size. It follows that $2|S_1| \ge \frac{\alpha}{3}$ and $|S_1| \ge \frac{\alpha}{6}$. Finally, $|S_2| = |\mathcal{P} - (S_0 + S_1)| \ge n - (|F_1| + \frac{\alpha}{3}) \ge \alpha - \frac{\alpha}{3} \ge \frac{\alpha}{6}$.
- $F_1 < \frac{\alpha}{3}$ and $F_2 < \frac{\alpha}{3}$. The proof is similar to the proof of the second case and is omitted.

Note that we do not consider $F_1 < \frac{\alpha}{3}$ and $F_2 \ge \frac{\alpha}{3}$ because $|F_1| \ge |F_2|$. This completes the proof. ∎

Now, we describe the quorum system. On S_i, $i \in \{0, 1, 2\}$, define a quorum system with load $O(\frac{1}{\sqrt{|S_i|}})$. Many such systems exist. One such system is the triangle lattice system [2]. In the triangle lattice system over S_i, we can choose $\sqrt{2|S_i|}$ quorums such that each processor belongs to exactly two quorums. If we choose each quorum with a probability $\frac{1}{\sqrt{2|S_i|}}$, it follows that the load of the triangle lattice system on S_i is at most $\frac{2}{\sqrt{2|S_i|}} = \sqrt{\frac{2}{|S_i|}}$. Define a quorum on $\mathcal{P}$ to be the union of three sets: one from the triangle lattice system on S_0, another from the triangle lattice system on S_1 and the third from the triangle lattice system on S_2.

Proposition 22: *The resulting system tolerates $\mathcal{F}$.*

Proof: In fact, any two quorums intersect in three points that belong to three distinct elements of $\mathcal{F}$ and, therefore, the intersection of two quorums does not belong to the union of two faulty sets. ∎

Proposition 23: *The load of the quorum system constructed above is $O(\frac{1}{\sqrt{\alpha}})$.*

Proof: On each S_i, we can choose $\sqrt{2|S_i|}$ quorums such that each processor belongs to exactly two quorums. It follows that there are $\sqrt{8|S_0||S_1||S_2|}$ quorums on $\mathcal{P}$ corresponding to all combinations of these quorums. Each processor in S_i belongs to $2\sqrt{4|S_{(i-1)mod3}||S_{(i+1)mod3}|}$. It follows that the load on a processor in S_i is $\frac{2\sqrt{4|S_{(i-1)mod3}||S_{(i+1)mod3}|}}{\sqrt{8|S_0||S_1||S_2|}} = \sqrt{\frac{2}{|S_i|}} \le \sqrt{\frac{2}{\alpha/6}} = \sqrt{\frac{12}{\alpha}} = O(\sqrt{\frac{1}{\alpha}})$. ∎

Corollary 24: *The bound of Proposition 15 is tight.*

Note that the system presented in this section is not an asynchronous byzantine quorum system.

7 General Construction of Byzantine Quorums

The results of the previous section suggest a simple general way to construct byzantine quorums. The construction yields asynchronous byzantine quorum systems.

Let $\mathcal{Q}$ be a quorum system on a set $\mathcal{P}$ of n processors. A byzantine quorum on a set $\mathcal{P}'$ of $n(2f+1)$ processors can be obtained as follows. Partition $\mathcal{P}'$ into $2f+1$ sets of n processors each. For each set $\mathcal{P}_i$ of the partition, let β_i be a bijection between $\mathcal{P}$ and $\mathcal{P}_i$. Define $\mathcal{Q}_i = \beta_i(\mathcal{Q})$. It is clear that $\mathcal{Q}_i$ is a quorum system on $\mathcal{P}_i$. Let $\mathcal{Q}' = \{Q | Q = \bigcup_{Q_{j_i} \in \mathcal{Q}_i, i=1,\ldots 2f+1} Q_{j_i}\}$. It is easy to show that $\mathcal{Q}'$ is a byzantine quorum system on $\mathcal{P}'$ that can tolerate up to f arbitrary failures. Also, it is easy to show that the resulting quorum has a load equal to the load of $\mathcal{Q}$ (note that the resulting quorum is on a set whose size is $2f+1$ times the size of $\mathcal{Q}$). Even though the method is simple, we have seen that it can be used to construct byzantine quorum systems with optimal load.

8 Ackowledgements

I would like to thank Dahlia Malkhi for many clarifications and helpful discussions. I would also like to thank the reviewers for many useful suggestions.

References

[1] 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.

[2] R. A. Bazzi. Planar Quorums In *Proceedings of the 10th International Workshop on Distributed Algorithms*, pages 251-268, October 1996.

[3] H. Garcia–Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):481–860, 1985.

[4] D. K. Gifford. Weighted Voting for Replicated Data *Proceeding of 7th ACM Symposium on Operating Systems Principles*, pages 150–162, December 1979.

[5] M. P. Herlihy. *Replication Methods for Abstract Data Types.* Ph.D. Thesis, Massachusetts Institute of Technology, 1984.

[6] T. Ibaraki and T. Kameda. A theory of Coteries: Mutual Exclusion in Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):779–749,1993.

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

[8] A. Kumar., M. Rabinovich, and R. Sinha. A performance study of general grid structures for replicated data. In *Proceedings of International Conference on Distributed Computing Systems*, pages 178–185, May, 1993.

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

[10] D. Malkhi and M. Reiter. Byzantine Quorum Systems. In *Proceedings of ACM 29th Symposium on Theory of Computing*, May, 1997.

[11] D. Malkhi, M. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. In *Proceedings of the Sixteenth ACM Symposium on Principles of Distributed Computing*, August, 1997.

[12] S. J. Mullender and P. M. B. Vitanyi. Distributed Match Making. *Algorithmica*, 3:367–391, 1992.

[13] M. L. Neilsen *Quorum Structures in Distributed Systems.* Ph.D. Thesis, Department of Computer and Information Sciences, Kansas State University, 1992.

[14] M. L. Neilsen and M. Mizuno. Decentralized Consensus Porotocols In *Proceedings of 10th International Phoenix Conference on Computing and Communications*, pages 257–262, 1991.

[15] 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.

[16] D. Peleg and A. Wool. How to be an Efficient Snoop, or the Probe Complexity of Quorum Systems. In *Proceedings of the 15th ACM Symposium on Principles of Distributed Computing*, pages 290-299, 1996.

[17] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210-223, 1995.

[18] D. Peleg and A. Wool. Crumbling Walls: A class of high availability quorum systems. In *Processedings 14th ACM Symposium on Principles of Distributed Computing*, pages 120–129, 1995.

[19] D. Skeen. A quorum–based commit protocol. In *Proceedings of 6th Berkeley Workshop on Distributed Data Management and Computer Networks*, pages 69–80, 1982.