Efficient Survivability for Highly Replicated Services

Michael G. Merideth

April 2009
CMU-ISR-09-112

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Michael K. Reiter, chair
Anupam Gupta
David R. O'Hallaron
Mustaque Ahamad, Georgia Institute of Technology

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

This work was partially supported by NSF grant CCF-0424422.

Keywords: Distributed systems, Byzantine fault tolerance, probabilistic quorum systems

Abstract

Networked services like distributed file systems can suffer a wide range of problems such as machine crashes and security intrusions that may cause downtime, incorrect behavior, and other undesirable issues. Byzantine-fault-tolerant protocols that replicate the services represent catchall solutions for such problems by providing survivability—the ability of a service to operate correctly despite instances of such security or reliability "faults". These protocols use Byzantine quorum systems as building blocks in order to ensure that services operate correctly and are available to clients even in the presence of faults. However, there are a variety of Byzantine quorum systems, which differ in three primary measures: fault tolerance—a measure of the number of faults that can be tolerated; load—a measure of efficiency; and availability—a measure of how well the service remains available for use. Unfortunately, no quorum system excels in all categories.

In this dissertation, we show that, compared with previous quorum systems, *probabilistic quorum systems* can provide better fault tolerance and load albeit at the cost of admitting a bounded probability of failing to mask faults. We present a probabilistic opaque quorum system that can tolerate up to 37% more faults than can traditional opaque quorum systems. Then, we present a technique called *write markers* for probabilistic masking and opaque quorum systems that can tolerate 50% and 48% more faults, respectively, compared with traditional quorum systems. Moreover, these probabilistic quorum systems with write markers have asymptotically better load than the bounds proven for previous masking and opaque quorum systems, achieving an optimal $O(1/\sqrt{n})$ in certain cases, and presenting the possibility of smaller quorums that yield efficient access for clients.

Additional contributions of this dissertation include the following. First, we present a framework for comparing Byzantine-fault-tolerant protocols on the basis of how they use quorum systems. This framework highlights the similarity between protocols that are explicitly quorum based, such as Q/U, and others like BFT and Zyzzyva that are not. Second, we introduce a way of improving the availability of probabilistic quorum systems by providing some leeway in how quorums are chosen. Instead of assuming that all quorums are chosen uniformly at random, we allow faulty clients to choose quorums by any strategy from access sets that are chosen uniformly at random. Third, we present the first analysis of a probabilistic quorum system that accounts for the behavior of Byzantine-faulty clients. We anticipate that a faulty client may choose quorums with the goal of maximizing the error probability, and show the effects that this may have. Fourth, we present a framework based on the McDiarmid inequality in order to prove that probabilistic quorum systems in general can meet specific load and fault tolerance targets. This framework allows us to prove that probabilistic masking quorum systems can tolerate up to 13% more faults than shown using Chernoff bounds previously. Fifth, we present a protocol by which probabilistic quorum systems can tolerate Byzantine-faulty clients. Such clients are otherwise problematic in that they may seek to cause the system to fail to mask faults. Finally, we analyze the cost of changing quorums routinely, as may be required by probabilistic quorum systems. This analysis is in the context of wide area networks with the Q/U protocol, which can require state to be transferred between servers as a result of quorum changes.

Acknowledgments

The following people deserve special mention. My wife, Iryna, for her loving care and support. My advisor, Mike Reiter, for his guidance, for providing an amazing research environment focused on quality, for North Carolina, and for all of his help and time. Dave O'Hallaron for Internet Services, for discussions and reviews, and for sitting on my thesis committee. Anupam Gupta and Mustaque Ahamad for their valuable comments and suggestions. David Garlan, Jim Herbsleb, Priya Narasimhan, Bill Scherlis, and the other faculty members. Mike, Paul, Roland, George, Charles, and all of the other students. Florian Oprea, Arun Iyengar, Thomas Mikalsen, Stefan Tai, and Isabelle Rouvellou for research collaborations. Patrick Sobalvarro, Jonathan Rees, Richard Kelsey, and Eric Chown for helping me to get started. To my family and friends, thank you so much for your support.

Contents

1 Introduction **1**
- 1.1 Byzantine-Fault-Tolerant Service Protocols . . . 1
  - 1.1.1 Example Application: Distributed File Storage . . . 2
  - 1.1.2 Consistency Protocols . . . 2
- 1.2 Byzantine Quorum Systems . . . 2
  - 1.2.1 Dissemination, Masking, and Opaque . . . 3
  - 1.2.2 Faults and Load . . . 4
  - 1.2.3 Probabilistic Quorum Systems . . . 4
  - 1.2.4 Write Markers for Probabilistic Quorum Systems . . . 5
- 1.3 Thesis Statement and Contributions . . . 6

2 Context: Byzantine Quorums in Protocols **9**
- 2.1 System Model for the Protocols . . . 9
- 2.2 Sizes of Sets . . . 9
- 2.3 Read-Overwrite Protocols . . . 11
  - 2.3.1 Use of Masking Quorums . . . 12
  - 2.3.2 Creating Self-Verifying Data . . . 13
- 2.4 State-Machine-Replication Protocols . . . 14
  - 2.4.1 The Framework . . . 15
  - 2.4.2 Use of Dissemination Quorums . . . 16
  - 2.4.3 Use of Opaque Quorums . . . 18
  - 2.4.4 Other Trade-Offs . . . 21
- 2.5 Summary . . . 21

3 System Model and Definitions **23**
- 3.1 Conventions . . . 23
- 3.2 Fault Model . . . 23
- 3.3 Service Semantics and Consistency Model . . . 23
- 3.4 Behavior of Benign Clients . . . 24
- 3.5 Voting . . . 24
- 3.6 Repair . . . 25
- 3.7 Access Strategy and Error . . . 25
- 3.8 Behavior of Faulty Clients . . . 25

3.9 Communication Assumptions 26

4 Basics of Probabilistic Quorum Systems 27
4.1 Consistency Properties 27
4.2 Behavior of Faulty Clients 27
4.2.1 Reads by a Faulty Client 28
4.2.2 Writes by a Faulty Client 28
4.3 Consistency Constraints 29
4.4 Summary 30

5 Framework for Proofs 31
5.1 McDiarmid Concentration Bound 31
5.2 General Bound for Probabilistic Quorums 32
5.3 Benefits Over Chernoff Bounds 35
5.4 Reevaluation: Probabilistic Dissemination and Masking Quorums 36
5.5 Summary 39

6 Probabilistic Opaque Quorum Systems 41
6.1 Consistency Constraints 41
6.2 Implied Bounds 42
6.3 Spectrum of System Sizes 44
6.4 Error Probabilities 47
6.5 Summary 49

7 Write Markers 51
7.1 The Technique 51
7.2 Consistency Constraints with Write Markers 52
7.2.1 Masking with Write Markers 52
7.2.2 Opaque with Write Markers 52
7.3 Bounds 52
7.4 Summary 55

8 Protocol to Enforce the Access Strategy 57
8.1 Design Constraints 57
8.2 Obtaining a VRV 58
8.3 Choosing an Access Set 59
8.4 Server Verification 60
8.5 Background Propagation 60
8.6 Summary 62

9 Implementation of Write Markers 63
9.1 Adapting the Access-Restriction Protocol 63
9.2 Implementation for Probabilistic Opaque Quorums 65
9.3 Implementation for Probabilistic Masking Quorums 65

9.4 Summary . . . 66

10 The Cost of State Transfer **67**
10.1 Avoiding State Transfer on WANs . . . 67
10.2 Evaluation Challenges . . . 68
10.2.1 Increased Clients with Latency . . . 68
10.2.2 Limited Emulated Bandwidth . . . 70
10.3 Evaluation . . . 70
10.4 Summary . . . 73

11 Related Work **75**
11.1 Probabilistic Quorum Systems. . . . 75
11.2 Opaque Quorum Systems. . . . 75
11.3 Signed Quorum Systems . . . 76
11.4 K-Quorums . . . 76
11.5 Tolerating Byzantine Clients . . . 76
11.6 Write Markers . . . 77
11.7 Other Scalability Approaches . . . 77
11.8 Distributed Hash Tables . . . 78

12 Conclusions **81**
12.1 Summary . . . 81
12.2 Future Work . . . 81

List of Figures

2.1 Phases for an overwrite in a read-overwrite protocol. . . . 11
2.2 The phases for a read in a read-overwrite protocol. . . . 11
2.3 The phases for an overwrite in the PASIS-RW protocol. . . . 12
2.4 The phases for a read in the PASIS-RW protocol. . . . 12
2.5 Making data self-verifying. . . . 13
2.6 The stages of Byzantine-quorum state-machine-replication protocols. . . . 15
2.7 BFT. . . . 17
2.8 Optimistic update versus optimistic accept. . . . 18
2.9 Zyzzyva. . . . 19
2.10 The Q/U protocol (optimistic accept and update). . . . 20
2.11 Optimistic accept and update in Q/U and FaB Paxos (both optimized). . . . 20

6.1 Sizes of access sets to achieve a given lower bound on n. . . . 46
6.2 Number of servers required to achieve given calculated worst-case error probability. . . . 48

9.1 Read operation with write markers. . . . 64
9.2 Message types. . . . 64
9.3 Write operation with write markers. . . . 66

10.1 Rate of requests from clients. . . . 69
10.2 Doubling of the traffic due to the emulator. . . . 70
10.3 Q/U with state transfer. . . . 72
10.4 Q/U with state transfer, no delay. . . . 72

List of Tables

1.1 Properties of quorum systems. 5

2.1 Threshold quorum systems. 22

6.1 Lower bounds on n for various configurations. 45

7.1 Ability of a server to vote for a given candidate: $\bullet$ (traditional quorums); $\star$ (write markers). . 51

10.1 Round trip network latency data (ms). 71

Chapter 1

Introduction

Distributed services can suffer a diverse range of problems including crashes of components, compromises of servers, bugs in software, bit flips in hardware, and unreliable networks. Two general categories of solutions are *prevention*, which makes the problems less likely, and *tolerance*, which makes the service work despite the problems. Examples of solutions for prevention include: components with better reliability; additional computer security measures; and software with fewer bugs. The use of redundant components to mask crashes is an example of a solution for tolerance. Today, through techniques tailored to the specific problem at hand, problems are often addressed on a case-by-case basis using one or more of these approaches.

However, addressing problems individually can be costly, or even intractable, given a sufficiently difficult or large set. For example, aspects of computer security can be viewed as an "arms race" where increasingly sophisticated security solutions are developed to thwart increasingly sophisticated attackers who are able to defeat earlier solutions. Users are frequently asked to patch their software when new bugs (which often lead to security vulnerabilities) are found in new (or not-so-new) software. Thus, preventing problems with computer security can be a process without end. For reasons such as this, finding a generic solution that can tolerate a wide range of problems is an attractive alternative.

1.1 Byzantine-Fault-Tolerant Service Protocols

Byzantine fault tolerance [37] is a way to tolerate many problems without regard as to their specifics. The Byzantine fault model does not specify the behavior of faulty components. Instead, it allows that a faulty component can behave arbitrarily. For example, in this fault model, both the omission of results and incorrect results are types of faults. Byzantine fault tolerance uses replication (i.e., redundancy) to mask the behavior of up to b faulty replicas out of the total of n replicas. As such, Byzantine fault tolerance provides a form of *survivability* [26]—the ability of the service to operate correctly despite intrusion or partial failure.

It should be noted that, for the full benefits of Byzantine fault tolerance, it must be ensured that no more than b replicas are faulty. This means, for example, that care must be taken so that a common vulnerability is not exploited in more than b replicas. For a survey of possibilities, see the "Future Work" section of [20]. Here, we do not provide additional insight into such solutions. Instead, we operate under the assumptions of our system model found in Chapter 3.

1.1.1 Example Application: Distributed File Storage

One common target application for Byzantine fault tolerance is a reliable data store like PASIS [29]. A data store maintains both file data and metadata (such as directory location and timestamp). File data is modified in a read-overwrite manner: one or more clients can overwrite the existing data in a file with new data; and one or more clients can read data from files. The file store is not concerned with the semantics of the file data; from its perspective, file data can be modified arbitrarily when overwritten. Metadata, however, should only be modified following precise protocols. For example, when a file is moved from one directory to another, multiple pieces of metadata must be modified together, or else the metadata system may become inconsistent. Logically, if only the destination directory is updated, then the file may now exist in two locations; if only the source directory is updated, the file may effectively be deleted. Thus, to ensure that the operations work correctly, the service itself may have to modify the metadata.

For reliability, a data store can be replicated such that redundant copies of both file data and metadata are maintained by different replicas. When the redundant copies are spread across multiple servers, the failure of a server can be tolerated because another copy still exists. However, as discussed next, the maintenance of the copies requires the use of a *consistency protocol* so that, for example, the surviving copy of the data is not out of date compared to the copy that is lost due to the failure.

1.1.2 Consistency Protocols

Because of the replication inherent in a Byzantine-fault-tolerant system, use of at least one consistency protocol is generally necessary. Some Byzantine-fault-tolerant services can provide to their clients a rich interface of arbitrary operations to be performed by the service. For example, in a data store, the metadata service provides operations such as the "move file" operation described above. In order to mask any potential inconsistency that comes from faulty replicas, each replica generally performs the operation locally. A *state-machine-replication protocol* [66] is used to ensure that the correct result is chosen by the client and that the system remains consistent.

In a state-machine-replication protocol, the replicas do more than accept reads and overwrites—they process updates based on the state they already hold. However, some services do not need the full generality of providing arbitrary operations performed by the service. For example, to maintain file data that is only read and overwritten, the file server needs to provide only read and overwrite operations. These services can generally use *read-overwrite consistency protocols*[1] such as [29, 18, 30] instead of the more complex state-machine-replication protocols.

1.2 Byzantine Quorum Systems

Consistency protocols that can tolerate Byzantine faults are often based on the use of quorum systems. Quorum systems are a fundamental building block for Byzantine-fault-tolerant protocols. In this dissertation, we focus on improving properties of Byzantine quorum systems through the use of probabilistic quorum systems. In this section, we provide background information on Byzantine quorum systems as context for the remainder of the dissertation. We begin by defining the three types of quorum systems with which we

[1] In other contexts, such protocols are often called read-write protocols. Here, we use the term overwrite to distinguish from writes that occur in quorum systems.

are concerned; we follow this with an introduction to probabilistic quorum systems; and, finally, discuss *write markers*, our technique for improving the scalability of probabilistic quorum systems.

1.2.1 Dissemination, Masking, and Opaque

A *quorum system* is a collection of sets (*quorums*) of servers, such that any two quorums have non-empty intersection. More precisely, given a universe U of servers, a quorum system over U is a collection $\mathcal{Q} = \{Q_1, \ldots, Q_m\}$ such that each $Q_i \subseteq U$ and

$$|Q \cap Q'| > 0 \tag{1.1}$$

for all $Q, Q' \in \mathcal{Q}$. Each Q_i is called a *quorum*. The intersection property (1.1) makes quorums a useful primitive for coordinating actions in a distributed system. For example, if clients perform writes at a quorum of servers, then a client who reads from a quorum will observe the last written value in the response from at least one server. Because of their utility in such applications, quorums have a long history in distributed computing, and are a standard tool for improving the efficiency and fault-tolerance of a distributed system.

In many respects, quorum systems are well-suited to wide-area networks in which network delays to reach servers can be varied and significant. In contrast to read-one, write-all (ROWA) replication schemes that require every server to be contacted, quorum systems can work without the participation of the most distant servers. In a typical quorum-based application, a read or write operation is performed by contacting all of the servers of some quorum (possibly in more than one round of interaction). An intersection property like (1.1) enables a read or write operation to observe the effects of prior write operations. Moreover, the fact that not all servers need to be contacted for each operation can increase efficiency if the processing load is dispersed across servers through the use of different quorums. These properties of quorum systems make them the tool of choice for a number of practical protocol implementations (e.g., [1, 22, 23, 29, 34, 44, 45, 48]).

In systems that may suffer Byzantine faults [37], the intersection property (1.1) is typically not adequate as a mechanism to enable consistent data access. Because (1.1) requires only that the intersection of quorums be non-empty, it could be that two quorums intersect only in a single server, for example. In a system in which up to $b > 0$ servers might suffer Byzantine faults, this single server might be faulty and, consequently, could fail to convey the last written value to a reader.

For this reason, Malkhi and Reiter [43] proposed various ways of strengthening the intersection property (1.1) so as to enable quorums to be used in Byzantine environments. If B is the (unknown) set of all (up to b) servers that are faulty, then the number of non-faulty servers in two quorums $Q, Q' \in \mathcal{Q}$ is

$$|(Q \cap Q') \setminus B|.$$

Drawing a parallel with (1.1), we have

$$|(Q \cap Q') \setminus B| > 0 \tag{1.2}$$

for all $Q, Q' \in \mathcal{Q}$. In other words, there is at least one non-faulty server in the intersection of any two quorums. Therefore, following the previous example, at least one (non-faulty) server will convey the last written value to a reader. These quorum systems are called *dissemination* systems.

A more stringent alternative to (1.1) is

$$|(Q \cap Q') \setminus B| > |Q' \cap B| \tag{1.3}$$

for all $Q, Q' \in \mathcal{Q}$. In other words, the intersection of any two quorums contains more non-faulty servers than the faulty ones in either quorum. As such, the responses from these non-faulty servers will outnumber those from faulty ones. These quorum systems are called *masking* systems.

Opaque quorum systems, have an even more stringent requirement as an alternative to (1.1):

$$|(Q \cap Q') \setminus B| > |(Q' \cap B) \cup (Q' \setminus Q)| \tag{1.4}$$

for all $Q, Q' \in \mathcal{Q}$. In other words, the number of non-faulty servers in the intersection of Q and Q' (i.e., $|Q \cap Q' \setminus B|$) exceeds the number of faulty servers in Q' (i.e., $|Q' \cap B|$) together with the number of servers in Q' but not Q. The rationale for this property can be seen by considering the servers in Q' but not Q as "outdated", in the sense that if Q was used to perform a write to the system, then those servers in $Q' \setminus Q$ are unaware of the write. As such, if the faulty servers in Q' behave as the outdated ones do, their behavior (i.e., their responses) will dominate that from the non-faulty servers in the intersection ($Q \cap Q' \setminus B$) unless (1.4) holds.

1.2.2 Faults and Load

The increasingly stringent properties of Byzantine quorum systems come with costs in terms of the smallest system sizes that can be supported while tolerating a number b of faults [43]. This implies that a system with a fixed number of servers can tolerate fewer faults when the property is more stringent as seen in Table 1.1, which refers to the quorums just discussed as *strict*. Table 1.1 also shows the negative impact on the ability of the system to disperse load amongst the replicas, as discussed next.

Naor and Wool [58] introduced the notion of an *access strategy* by which clients select quorums to access. An access strategy $p : \mathcal{Q} \rightarrow [0, 1]$ is simply a probability distribution on quorums, i.e., $\sum_{Q \in \mathcal{Q}} p(Q) = 1$. Intuitively, when a client accesses the system, it does so at a quorum selected randomly according to the distribution p.

The formalization of an access strategy is useful as a tool for discussing the load-dispersing properties of quorums. The *load* [58] of a quorum system, $\mathcal{L}(\mathcal{Q})$, is the probability with which the busiest server is accessed in a client access, under the best possible access strategy p. As listed in Table 1.1, tight lower bounds have been proven for the load of each type of strict Byzantine quorum system. The load for opaque quorum systems is particularly unfortunate—systems that utilize opaque quorum systems cannot effectively disperse processing load across more servers (i.e., by increasing n) because the load is at least a constant. Byzantine quorum systems are used by many modern Byzantine-fault-tolerant protocols, e.g., [1, 22, 29, 34, 44, 48] in order to tolerate the arbitrary failure of a subset of their replicas. As such, circumventing the bounds is an important topic.

1.2.3 Probabilistic Quorum Systems

One way to circumvent these bounds is with *probabilistic quorum systems* [47]. Probabilistic quorum systems relax the quorum intersection properties, asking them to hold only with high probability. More specifically, they relax (1.2), (1.3), or (1.4), for example, to hold only with probability $1 - \epsilon$ (for ϵ, a small

constant), where probabilities are taken with respect to the selection of quorums according to an access strategy p [47, 53]. This technique has been shown to yield dissemination quorums tolerating $b < n$ and masking quorum systems tolerating $b < n/3$. With the proof framework from Chapter 5 we show that probabilistic masking quorum systems can actually do somewhat better than this, tolerating $b < n/2.62$. Moreover, in Chapter 6 we present a probabilistic opaque quorum system that can tolerate $b < n/3.15$ as seen in Table 1.1. These bounds hold in the sense that for any $\epsilon > 0$ there is an n_0 such that for all $n > n_0$, the required intersection property ((1.2), (1.3), or (1.4) for dissemination, masking, and opaque quorum systems, respectively) holds with probability at least $1 - \epsilon$. Unfortunately, except in the case of dissemination quorums, probabilistic quorum systems alone do not materially improve the load of Byzantine quorum systems.

The probability of error in probabilistic quorums requires mechanisms to ensure that accesses are performed according to the required access strategy p if the clients cannot be trusted to do so. In Chapter 8 we provide a technique to coerce all clients to follow the access strategy.

1.2.4 Write Markers for Probabilistic Quorum Systems

As one contribution, we present an additional modification, *write markers*, that improves further on the bounds of masking and opaque probabilistic quorum systems. Intuitively, in each write access to a quorum of servers, a write marker is placed at the accessed servers in order to evidence the quorum used in that access. This write marker identifies the quorum used; as such, faulty servers not in this quorum cannot respond to subsequent quorum accesses as though they were.

Table 1.1: Contrasting properties of strict quorum systems with those of probabilistic quorum systems with and without write markers. (**Bold** entries are properties of particular constructions; others are lower bounds.)

Non-Byzantine:	**load**		**faults**	
strict	$\Omega(1/\sqrt{n})$	[58]	$< n$	
Dissemination:	**load**		**faults**	
strict	$\Omega(\sqrt{b/n})$	[43]	$< n/3.00$	[46]
probabilistic	$\mathbf{O(1/\sqrt{n})}$	[47]	$< \mathbf{n}$	[47]
Masking:	**load**		**faults**	
strict	$\Omega(\sqrt{b/n})$	[43]	$< n/4.00$	[46]
probabilistic	$\Omega(b/n)$	[47]	$< \mathbf{n/2.62}$	[here]
write markers	$\mathbf{O(1/\sqrt{n})}$	[here]	$< \mathbf{n/2.00}$	[here]
Opaque:	**load**		**faults**	
strict	$\geq 1/2$	[43]	$< n/5.00$	[43]
probabilistic	unproven		$< \mathbf{n/3.15}$	[here]
write markers	$\mathbf{O(b/n)}$	[here]	$< \mathbf{n/2.62}$	[here]

As seen in Table 1.1, by using this method to constrain how faulty servers can collaborate, we show that probabilistic masking quorum systems with load $O(1/\sqrt{n})$ can be achieved, allowing the systems to disperse load independently of the value of b. Further, probabilistic opaque quorum systems with load $O(b/n)$ can be achieved, breaking the constant lower bound on load for opaque systems. Moreover, the resilience of probabilistic masking quorums can be improved an additional 24% to $b < n/2$, and the resilience of probabilistic opaque quorum systems can be improved an additional 17% to $b < n/2.62$. In Chapter 9, we adapt the access-restriction protocol of Chapter 8 to accommodate write markers. With this, we provide a protocol to implement write markers that tolerates Byzantine clients.

Our analysis of write markers yields the following results:

Masking Quorums: We show that the use of write markers allows probabilistic masking quorum systems to tolerate up to $b < n/2$ faults when quorums are of size $\Omega(\sqrt{n})$. Setting all quorums to size $\rho\sqrt{n}$ for some constant ρ, we achieve a load that is asymptotically optimal for any quorum system, i.e., $\rho\sqrt{n}/n = O(1/\sqrt{n})$ [58]. This represents an improvement in load and the number of faults that can be tolerated. Probabilistic masking quorums without write markers can tolerate up to $b < n/2.62$ faults [53] and achieve load no better than $\Omega(b/n)$ [47]. In addition, the maximum number of faults that can be tolerated is tied to the size of quorums [47]. Thus, without write markers, achieving optimal load requires tolerating fewer faults. Strict masking quorum systems can tolerate (only) up to $b < n/4$ faults [43] and can achieve load $\Omega(\sqrt{b/n})$ [46].

Opaque Quorums: We show that the use of write markers allows probabilistic opaque quorum systems to tolerate up to $b < n/2.62$ faults. We present a construction with load $O(b/n)$ when $b = \Omega(\sqrt{n})$, thereby breaking the constant lower bound of $1/2$ on the load of strict opaque quorum systems [43]. Moreover, if $b = O(\sqrt{n})$, we can set all quorums to size $\rho\sqrt{n}$ for some constant ρ, in order to achieve a load that is asymptotically optimal for any quorum system, i.e., $\rho\sqrt{n}/n = O(1/\sqrt{n})$ [58].

This represents an improvement in load and the number of faults that can be tolerated. Probabilistic opaque quorum systems without write markers can tolerate (only) up to $b < n/3.15$ faults [53]. Strict opaque quorum systems can tolerate (only) up to $b < n/5$ faults [43]; these quorum systems can do no better than constant load even if $b = 0$ [43].

1.3 Thesis Statement and Contributions

Given a consistency protocol, if we wish to be able to tolerate more faults, i.e., increase b, then we typically must scale-up the number of replicas, i.e., n. Unfortunately, this increase in system size causes performance overheads because the quorums that must be contacted are larger. To address this issue, we expand the body of knowledge of probabilistic quorum systems as a type of quorum system that can allow for greater fault-tolerance and scalability. This leads to the following thesis statement.

We can increase both the degree of survivability and the scalability of quorum systems that underlie many Byzantine-fault-tolerant protocols at the cost of: (i) a bounded probability of inconsistency; and (ii) any overheads associated with frequent quorum changes.

Our contributions expand on those in four papers [53, 54, 55, 52] and are as follows:

- We present a technique called *write markers* for probabilistic masking and opaque quorum systems that can tolerate 50% and 48% more faults, respectively, compared with traditional quorum systems. Moreover, these probabilistic quorum systems with write markers have asymptotically better load than the bounds proven for previous masking and opaque quorum systems, achieving an optimal $O(1/\sqrt{n})$ in at least some cases.

- We present a probabilistic opaque quorum system without write markers that can tolerate up to 37% more faults than can traditional opaque quorum systems.

- We present a framework for comparing Byzantine-fault-tolerant protocols on the basis of how they use quorum systems. This framework highlights the similarity between protocols that are explicitly quorum based, such as Q/U [1], and others like BFT [22] and Zyzzyva [35] that are not.
- We present a framework based on the McDiarmid inequality [57] in order to prove that probabilistic quorum systems in general can meet specific load and fault tolerance targets. This framework allows us to prove that probabilistic masking quorum systems can tolerate up to 13% more faults than shown using Chernoff bounds previously [47].
- We introduce a way of improving the availability of probabilistic quorum systems by providing some leeway in how quorums are chosen. Instead of assuming that all quorums are chosen uniformly at random, we allow faulty clients to choose quorums by any strategy from access sets that are chosen uniformly at random.
- We present the first analysis of a probabilistic quorum system that accounts for the behavior of Byzantine-faulty clients. We anticipate that a faulty client may choose quorums with the goal of maximizing the error probability, and show the effects that this may have.
- We present a protocol by which probabilistic quorum systems can tolerate Byzantine-faulty clients. Such clients are otherwise problematic in that they may seek to cause the system to fail to mask faults.
- We analyze the cost of changing quorums routinely, as may be required by probabilistic quorum systems. This analysis is in the context of wide area networks with the Q/U protocol, which can require state to be transferred between servers as a result of quorum changes.

Chapter 2

Context: Byzantine Quorums in Protocols

In this chapter we seek to familiarize the reader further concerning dissemination, masking, and opaque quorum systems. For this purpose, we present two related frameworks. The frameworks highlight the characteristics of different Byzantine quorum systems via their use in protocols. For each quorum system, we provide, in the context of a framework, a description of at least one protocol that uses the quorum system. Our framework for overwrite operations for read-overwrite protocols consists of three high-level phases: propose; accept; and learn. Our framework for state-machine-replication protocols builds on this, adding a second stage to the learn phase, and adding a final verify phase. The frameworks allow us to highlight the differences in the protocols as they relate to the quorum systems. We consider the numbers of servers required, the numbers of faults that can be tolerated, and the numbers of rounds of communication required.

2.1 System Model for the Protocols

The system consists of a universe U of n replicas, and an arbitrary, but bounded, number of clients. There is a set $B \subset U$ that represents the b faulty replicas; the composition of B is known by the faulty clients and replicas, but not by the non-faulty ones. The remaining $n - b$ replicas, i.e., $U \setminus B$, are non-faulty. Faulty replicas and clients can behave arbitrarily (i.e., Byzantine faults [37]), but, as is typical, are computationally bound such that, e.g., they cannot subvert cryptographic primitives (e.g., cryptographic hash functions) used in the protocols. The protocols maintain consistency without assumptions about the processing rates of non-faulty clients and replicas or the message delays of the network (asynchronous timing model). However, for availability, these delays must be finite, and the processing rates must be non-zero.

2.2 Sizes of Sets

The meanings of quantities such as $n - b$, $b + 1$, and $2b + 1$ can be a source of confusion. This is sometimes compounded in descriptions of protocols by substitution of quantities that are not necessarily equivalent in all contexts. For example, the quantity $2b + 1$ has been used for at least three distinct conceptual purposes (described in more detail below): (i) the maximum number of responses for which to wait given the asynchronous timing model ($n - b$); (ii) a set with a non-faulty majority ($2b + 1$); and (iii) the set size that

guarantees at least one non-faulty replica in the intersection of any two sets ($\lceil (n+b+1)/2 \rceil$). As seen in the following equations, these quantities are equivalent under the assumption that $n = 3b + 1$:

$$\begin{aligned}
&\lceil (n+b+1)/2 \rceil \\
&\quad = \lceil (3b+1+b+1)/2 \rceil \\
&\quad = \lceil (4b+2)/2 \rceil \\
&\quad = (2b+1). \\
&(n-b) \\
&\quad = (3b+1-b) \\
&\quad = (2b+1).
\end{aligned}$$

However, these quantities are not necessarily equivalent given other values for n or other types of quorum systems. Note in particular that $n - b$ and $\lceil (n+b+1)/2 \rceil$ depend on the values of n and b while $2b + 1$ depends only on b. Because these quantities are very different conceptually, the use of a single identifier like $2b + 1$ can make the description of a protocol difficult to decipher.

Guaranteed responses ($n - b$). Given an asynchronous timing model and a system that can tolerate up to b faults, a process can wait for up to, but not more than, $n - b$ responses. This is because b replicas may be faulty and may never respond, and it is generally impossible to distinguish between a faulty process and one that is slow. However, it may be possible to wait for more than $n - b$ responses in specific cases such as when it can be determined that the set of responses received contains at least a certain number from faulty replicas.

Guaranteed non-faulty responses ($n - 2b$). As described above, one can wait for only $n - b$ responses. Of these responses, b may be from faulty replicas—the b replicas not represented in the set of responses may have simply been slower.

At least one non-faulty replica ($b + 1$). Because there are at most b faulty replicas, any set of $b + 1$ responses contains at least one response from a non-faulty replica.

Non-faulty majority ($2b + 1$). Because there are at most b faulty replicas, any set of $2b + 1$ replicas necessarily contains at least $b + 1$ non-faulty replicas. As such, the majority of any set of at least $2b + 1$ replicas is non-faulty.

Replica in intersection ($\lceil (n+1)/2 \rceil$). Any two sets of $\lceil (n+1)/2 \rceil$ replicas chosen from the n total replicas contain at least one replica in common.

Non-faulty replica in intersection ($\lceil (n+b+1)/2 \rceil$). Any two sets of $\lceil (n+b+1)/2 \rceil$ replicas chosen from the n total replicas contain at least one non-faulty replica (i.e., at least $b + 1$ replicas) in common.

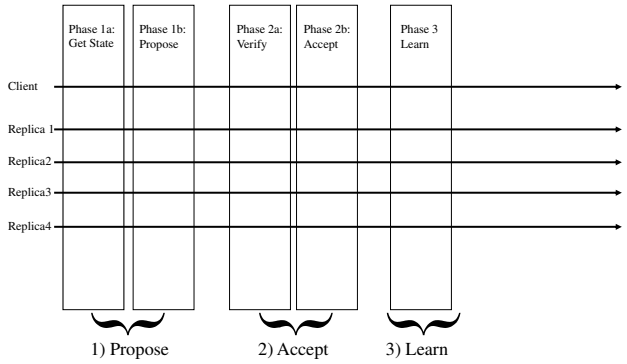


Figure 2.1: Phases for an overwrite in a read-overwrite protocol.

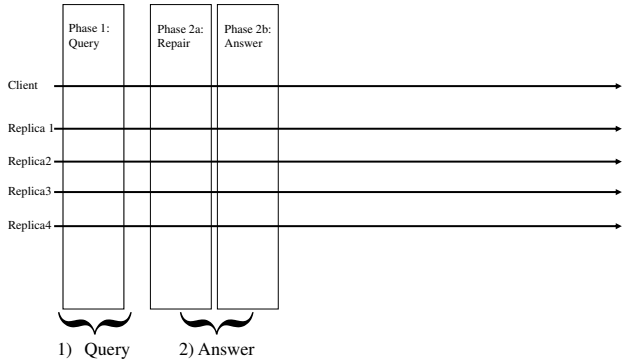


Figure 2.2: The phases for a read in a read-overwrite protocol.

$c+1$ **non-faulty replicas in intersection ($\lceil (n+b+c+1)/2 \rceil$).** For any non-negative constant c, any two sets of $\lceil (n+b+c+1)/2 \rceil$ replicas chosen from the n total replicas contain at least $c+1$ non-faulty replicas (i.e., at least $b+c+1$ replicas) in common.

2.3 Read-Overwrite Protocols

Figure 2.1 shows the phases for an overwrite operation in a typical read-overwrite protocol. The first phase, propose, is where the client submits the operation. In some protocols, such as the PASIS-RW protocol [29] described below, the client must first determine the current state of the system, e.g., in order to determine what the next sequence number should be. If necessary, this is done in phase 1a. In phase 1b, the client sends the operation to at least a quorum of servers. In phase 2, servers perform any necessary verification, and accept the operation. In phase 3, the client knows that the operation is complete once it has been accepted by a quorum of servers.

Figure 2.2 shows the phases for a read operation in a typical read-overwrite protocol. To perform an operation, a client contacts at least a quorum of servers in phase 1. For the service to provide linearizable semantics [31], the client must be certain that the value it reads is the result of a complete overwrite. If not, then another client performing a later read might read an earlier or different value. Therefore, if the client is uncertain whether the overwrite is complete, the client repairs it, e.g., by completing it at a quorum of

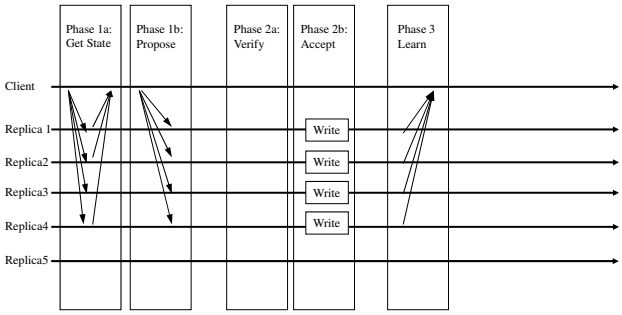


Figure 2.3: The phases for an overwrite in the PASIS-RW protocol.

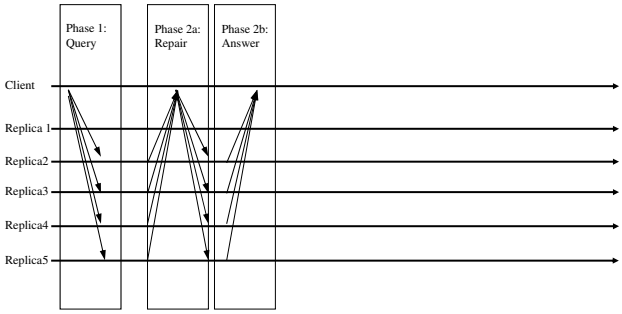


Figure 2.4: The phases for a read in the PASIS-RW protocol.

servers in phase 2a. Finally, if the client receives the same value from at least a quorum of servers in phase 2b, then it knows the overwrite is complete and uses this value as the result of the read operation.

2.3.1 Use of Masking Quorums

To make this more concrete, consider the PASIS-RW protocol [29]. Figures 2.3 and 2.4 show a simplified view of the protocol that omits details such as erasure coding that are discussed in [29].

The PASIS-RW protocol uses a masking quorum system that satisfies (1.3). In an overwrite operation, the client first determines the most recent timestamp by querying at least a quorum in phase 1a. Let us assume that the system does not need repair. Then, based on the retrieved state and details such as the client identification number and the request description, the client generates a unique timestamp that is greater than those returned by the quorum. In PASIS-RW, timestamps must be strictly increasing so as to identify the most recent overwrite, but need not be consecutive. This fact allows timestamps to be ordered partially by the identification number of the client, and therefore to be generated without involvement of the replicas.

Using this new timestamp, the client submits the overwrite in phase 1b. A non-faulty server accepts the overwrite if it has not yet accepted one that is more recent, storing it in phase 2b. In phase 3, the client considers the operation complete upon learning that it has been accepted by at least a quorum of servers. The protocol includes mechanisms not discussed here for handling the case where the client does not receive such notification in phase 3.

In a read operation, the client submits the read request to at least a quorum of servers. If the most recent overwrite is complete, the reading client will observe it from more than b servers. Consider the example in

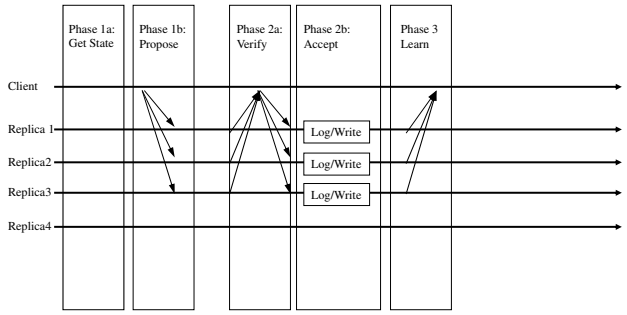


Figure 2.5: Making data self-verifying.

the figures, and assume that replica number 4 is faulty. The first client used the quorum containing replicas numbered 1 through 4 for the overwrite. The reading client queries replicas 2 through 5. Replica 5 returns a previous value, and replica 4 might forge a newer value in an attempt to hide the overwrite. However, replicas 2 and 3 each return the correct value. Since the value is returned by two replicas, which is more than b but fewer than a quorum, the client repairs the overwrite using a quorum in phase 2a. Upon learning that the value has been accepted by a quorum, the client uses the value as the result of the read in phase 2b.

2.3.2 Creating Self-Verifying Data

The PASIS-RW protocol just discussed uses a masking quorum system to ensure that any values generated by faulty replicas are not observed in other quorums. If the data were self-verifying, a dissemination quorum system could be used instead because replicas would be unable to generate verifiable values at all. One way to make data self-verifying is to have each client use a digital signature scheme to sign each overwrite. Unfortunately, clients may suffer Byzantine faults in our fault model, and therefore they cannot be trusted.

One way to make an operation self-verifying without trusting clients involves using signatures from a quorum of replicas. Liskov and Rodrigues [40] show how to make operations self-verifying in a read-overwrite protocol with Byzantine clients. There are two steps: an echo protocol like that of Rampart [61] so that non-faulty servers known that other non-faulty servers are not accepting conflicting operations; and a way for clients to verify this as well during reads. For the echo protocol, the first step is to have a quorum of servers provide their signatures stating that they have *tentatively* accepted the operation. If a replica is willing to accept the operation upon the condition that other non-faulty replicas accept no conflicting operation, the replica sends a tentative accept (echo) response. Non-faulty servers only accept operations that have been tentatively accepted by a quorum of servers. Therefore, clients must send a quorum of signed echoes in a second phase; in Figure 2.5, this occurs in phase 2a. Servers log the quorum of signatures along with the operation in phase 2b.

The second part of making the operation self-verifying is that the servers return the quorum of signatures to the clients during read operations. A quorum of echo responses proves that no quorum will accept a conflicting operation. This is because every two quorums overlap in some positive number of non-faulty replicas, and no non-faulty replica sends echo messages for conflicting operations. By themselves, faulty servers are too few to generate a quorum of signatures that can be verified. During a read, a client verifies the signatures before using the value, and therefore, each overwrite is self-verifying.

2.4 State-Machine-Replication Protocols

State-machine-replication protocols must assign a total order to requests. To minimize the amount of communication between servers, protocols like Q/U [1] and FaB Paxos [48] use opaque quorum systems [43] to order requests *optimistically*. That is, servers independently choose an ordering, without steps that would be required to reach agreement with other servers; the steps are performed only if servers choose different orderings. Under the assumption that servers independently typically choose the same ordering, the optimistic approach can provide lower overhead in the common case than protocols like BFT [22], which require that servers perform steps to agree upon an ordering *before* choosing it [1]. However, optimistic protocols have the disadvantage of requiring at least $5b + 1$ servers to tolerate b server faults, instead of as few as $3b + 1$ servers, and so they cannot tolerate as many faults for a given number of servers.

State-machine-replication protocols assign a total order to requests as follows. Each request is assigned a *permanent sequence number* that exists from the time of assignment through the life of the system and is never changed.[1] We use the term sequence number to indicate that there is a totally-ordered chain of requests; however, the sequence number might be implemented as a logical timestamp [1] or other suitable device. Each permanent sequence number is assigned to a *single* request. Therefore, due to the Byzantine fault model, permanent sequence numbers cannot be assigned by a single replica or client, which might assign the same permanent sequence number to multiple requests.

In order to get a permanent sequence number, a request is first assigned a *proposed sequence number*. Unlike permanent sequence numbers, the same proposed sequence number may be assigned to multiple requests. Therefore, a proposed sequence number can be selected by a single client or replica in isolation.

A quorum system is used for the assignment of permanent sequence numbers. A proposed sequence number for the request is written to the quorum system made up of the replicas. Each non-faulty replica accepts the proposed sequence number only if it has not already assigned the sequence number to a different request. A sequence number is permanent if and only if it has been accepted by a quorum of replicas. This ensures that each permanent sequence number is assigned only to a single request.

The type of quorum system used for accepting proposed sequence numbers to make them permanent implies a lower bound on n in terms of b as discussed in Chapter 1.2. For example, an opaque quorum system requires at least $5b + 1$ replicas, but can accept a proposed sequence number in a single round of communication. On the other hand, dissemination and masking quorum systems need only $3b+1$ and $4b+1$ replicas but require more rounds.

A non-faulty replica executes a request only after all lower sequence numbers are assigned permanently and it has executed their corresponding requests. Individual replicas send responses to the client upon executing the request. If a non-faulty replica is waiting to execute a request because it is unaware of the assignment of an earlier sequence number, action is taken so that the replica obtains the missing assignment.

The client determines the correct result from the set of responses it receives by determining that the result is due to a permanent sequence number assignment and from at least one non-faulty replica. This works because each non-faulty replica that executes a request with a permanent sequence number returns the same, correct result to the client. However, faulty replicas, as well as non-faulty replicas that execute requests without permanent sequence numbers (an optimization employed by some protocols), may return incorrect results.

[1] We choose the passive voice in this description because details such as which clients/replicas are involved in assigning the sequence number are protocol-specific.

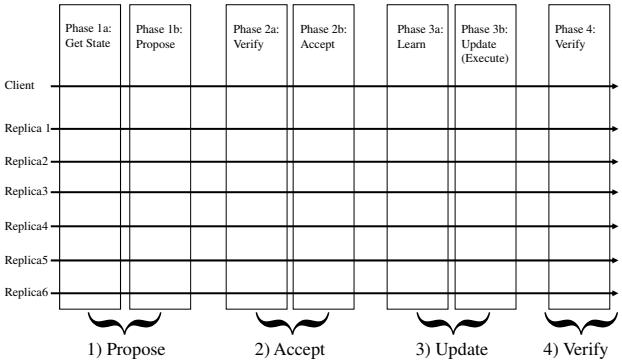


Figure 2.6: The stages of Byzantine-quorum state-machine-replication protocols.

Because of the use of the quorum system for sequence number assignments, none of the protocols surveyed become inconsistent in the face of a Byzantine-faulty proposer. However, the protocols each require a *repair* phase in order to continue to be able to make progress. The processes performing repair read from the quorum system to ensure that no permanent sequence number assignments are lost or changed. Repair is discussed further in Section 2.4.4.

2.4.1 The Framework

The framework depicted in Figure 2.6 is an extension of that in Figure 2.1. It consists of four high-level phases totaling seven sub-phases. In phase 1, a proposed sequence number is chosen for the client's request and sent to (at least) a quorum of replicas. In phase 2, a quorum of replicas accepts the proposed sequence number. If no quorum accepts the proposed sequence number assignment, a new proposal must be tried and the system may require repair (see Section 2.4.4). In phase 3, the request is executed according to the sequence number, and in phase 4 the client chooses the correct result. Phases 1a, 2a and 3a can be viewed as optional, as they are omitted by some protocols; however, omitting them has implications as discussed below. The remainder of this section explores each of the phases of the framework in greater detail.

Phase 1: Propose. Phase 1 is where a proposed sequence number is selected for a client request; this is done by a *proposer*, which, dependent on the protocol, is either a replica or a client. In some protocols, it is possible that the state of the system has been updated without the knowledge of the proposer (for example due to contention by multiple proposers). In this case the proposer may first need to retrieve the up-to-date state of the system, including earlier permanent sequence number assignments. This is the purpose of phase 1a. Phase 1b is where the proposed sequence number and request are sent to (at least) a quorum of replicas.

Phase 2: Accept. Phase 2 is where the proposed sequence number is either accepted or rejected. Depending on the type of quorum system, this may require a round of communication (corresponding to an echo phase as discussed in Section 2.3.2) for the purpose of ensuring that non-faulty replicas do not accept different conflicting proposals for the same sequence-number. If it requires this round of communication

(phase 2a), the protocol is said to employ a *pessimistic accept* phase, otherwise, it is said to employ an *optimistic accept* phase. In phase 2b, the sequence number assignment becomes permanent.

The primary benefit of an optimistic accept phase is that one round of communication (phase 2a) involving at least a quorum of replicas is avoided. The disadvantage is the need for an opaque quorum system, which requires $n > 5b$. The Zyzzyva protocol [35] discussed below avoids phase 2a in some executions (e.g., when the servers are all non-faulty and messages are delivered in a timely fashion).

Phase 3: Update. Phase 3 is where the update is applied, typically resulting in the execution of the requested operation. Like phase 2, this phase can be either *pessimistic* (requiring phase 3a), or *optimistic* (omitting phase 3a). Comparing phase 3a to the third phase of an overwrite operation in a read-overwrite protocol described in Section 2.3, we see that phase 3a allows the execution replicas to learn that the sequence number assignment has become permanent before performing the update. If phase 3a is omitted, the sequence number assignment may change and the request may need to be executed again with the permanent sequence number.

Since an optimistic update phase requires no additional round of communication before execution, it can lead to better performance. The disadvantages are that, as described below, clients must wait for a quorum of responses instead of just $b + 1$ to ensure that the sequence number assignment is permanent, and that computation may be wasted in the case that the proposed sequence number does not become permanent.

Phase 4: Verify. Phase 4 is where the client receives a set of responses. The client must verify that the update was based on a permanent sequence number assignment and performed by at least one non-faulty replica (in order to ensure that the result is correct). In general, this requires waiting for a quorum of identical responses indicating the sequence number, where the size of the quorum is dependent on the quorum system construction. However, if phase 3 is pessimistic, then no non-faulty replica will execute an operation unless the assignment is permanent. In this case, the client can rely on non-faulty replicas to verify that the sequence number is permanent, and so clients need wait for only $b + 1$ identical responses.

2.4.2 Use of Dissemination Quorums

Pessimistic Accept and Update—BFT (without optimizations).

BFT [22] is an example of a Byzantine-fault-tolerant state-machine-replication protocol that employs a pessimistic accept phase (with a dissemination quorum system) and a pessimistic update phase. As such, it involves communication in all four phases of our framework. Figure 2.7 shows the phases of an update request of the BFT protocol in the fault-free case. The operation is very similar to a protocol presented by Bracha and Toueg [16] also used by other protocols, e.g., [63, 17, 71, 36].

In the common case, there is a single proposer (called the *primary*) that itself is a replica; therefore, phase 1a is unnecessary—the proposer already knows the next unused sequence number. In phase 1b, the proposer unilaterally chooses a proposed sequence number for the request (a non-faulty proposer should choose the next unassigned sequence number) and writes (in the sense of a quorum system write, not an overwrite) the request along with the proposal to the other replicas in a message called PRE-PREPARE.

The verification done in phase 2a is equivalent to an echo protocol (though the responses are sent directly to all other replicas instead of through the proposer). This guarantees that non-faulty replicas do not accept different updates with the same proposed sequence numbers. Each replica other than the proposer sends a

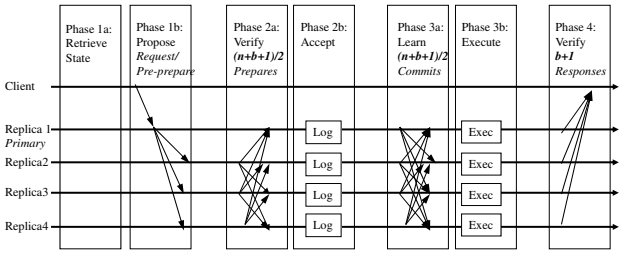


Figure 2.7: BFT.

PREPARE (i.e., echo) message including the proposal to all other replicas. If a replica obtains a quorum of matching PREPARE and PRE-PREPARE messages (including its own), it is guaranteed that no non-faulty replica will accept a proposal for the same sequence number but with a different request. Such a replica is called *prepared*. A prepared replica accepts the request in phase 2b, and logs the quorum of PREPARE and PRE-PREPARE messages as a form of proof. The sequence number assignment is permanent if and only if a quorum of replicas accepts the proposed sequence number in phase 2b.

In phase 3a, prepared replicas send COMMIT messages to all other replicas. A COMMIT message includes the quorum of PREPARE and PRE-PREPARE messages (i.e., echos) so that the sequence number assignment is self-verifying as discussed in Section 2.3.2. Because the update phase is pessimistic, in phase 3a, replicas wait to receive a quorum of COMMIT messages to make certain that the sequence number assignment is permanent. Having received a quorum of matching COMMIT messages for sequence number i, a replica executes the request only after executing all requests corresponding to permanent sequence number assignments $1 \,..\, i-1$.

Since BFT employs a pessimistic update phase, the client waits for only $b+1$ identical results in phase 4.

Pessimistic Accept, Optimistic Update—BFT w/ Tentative Execution.

One way to avoid a round of communication is to employ an optimistic update phase (i.e., to skip phase 3a). Castro and Liskov [22] detail an optimistic update optimization for BFT called tentative execution (TE). In tentative execution, phase 3a is omitted; however, the dissemination quorum system used to accept sequence numbers in phase 2 remains the same. Compared with unoptimized BFT, tentative execution saves a round of communication. However, since a response from a non-faulty replica no longer necessarily corresponds to a permanent sequence number assignment, the client must wait for a quorum of identical responses in phase 4 in order to ensure that the sequence number assignment is indeed permanent. In addition, replicas that execute a request corresponding to a non-permanent sequence number assignment that later changes (e.g., due to repair) may need to re-execute the request later.

Figure 2.8 shows the stages of the BFT protocol with the tentative-execution optimization, compared with the FaB Paxos protocol [48] that is discussed below. Note the smaller quorum size of BFT due to the use of a dissemination quorum system rather than the opaque quorum system required by FaB Paxos.

Speculative Accept, Optimistic Update—Zyzzyva.

Zyzzyva [35], shown in Figure 2.9, also uses tentative execution but can save an additional round of communication in "gracious" executions by additionally omitting phase 2a in such executions. Instead of waiting

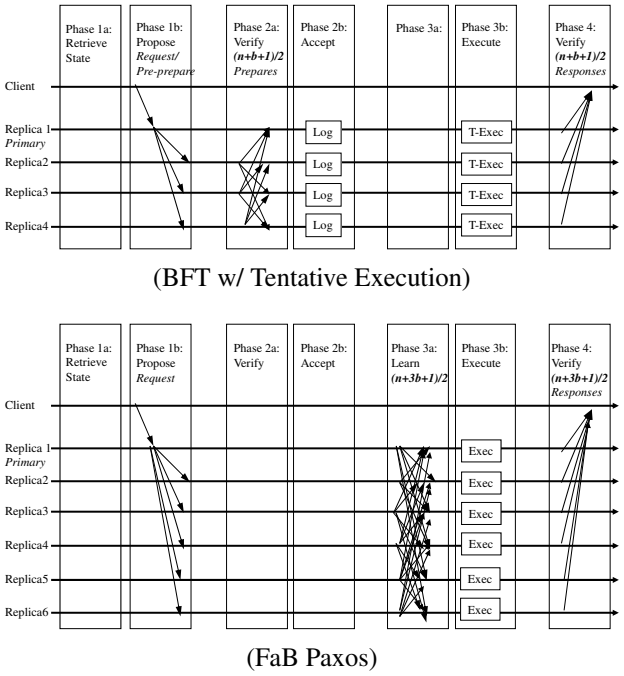


Figure 2.8: Optimistic update (BFT w/ tentative-execution optimization), compared to optimistic accept (FaB Paxos).

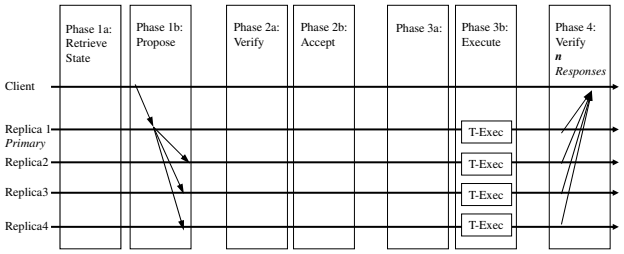
for a quorum of identical responses in phase 4, the client attempts to retrieve identical responses from all replicas. If successful, the client knows that, in any quorum, all of the non-faulty replicas in that quorum (i.e., at least $b + 1$ non-faulty replicas) will vouch for the sequence number assignment. Since these replicas are guaranteed to outnumber the faulty replicas, the sequence number assignment does not need to be self-verifying.

If the client does not receive identical responses from all servers, the execution is not gracious. In this case, phase 2a is necessary for the reasons described previously. If the client has received identical responses from at least a quorum of servers, the client executes phase 2a by sending the quorum of identical responses to the servers that each log this quorum of responses. If the client receives confirmation that at least a quorum of servers has logged this proof, then it knows that the sequence number assignment is self-verifying. Given this knowledge, the client accepts the result in phase 4.

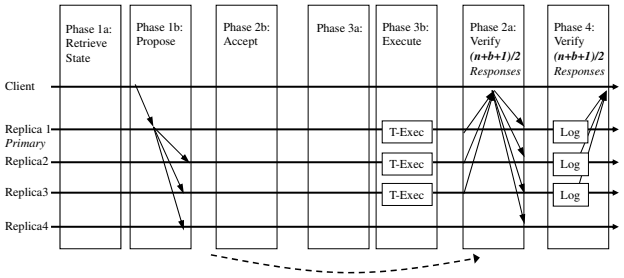
2.4.3 Use of Opaque Quorums

Optimistic Accept, Pessimistic Update—FaB Paxos.

Another way to avoid a round of communication is to employ an optimistic accept phase (i.e., to skip phase 2a). In relation to our framework, FaB Paxos [48], can be viewed as BFT with an optimistic accept phase (provided by the use of an opaque quorum system). It is seen in the lower half of Figure 2.8. Compared with BFT, FaB Paxos must use larger quorums, and therefore requires more replicas, in order to save this round of communication. While BFT with tentative execution and FaB Paxos (without tentative execution)



(Zyzzyva in Gracious Execution)



(Zyzzyva in Non-Gracious Execution)

Figure 2.9: Zyzzyva.

require the same number of message delays, they have different properties. In BFT, the execution may need to be rolled back and redone if a different sequence number assignment becomes permanent. In FaB Paxos, the execution is not tentative. However, because accept is optimistic, individual servers have no proof that the sequence number assignment is permanent, and so an opaque quorum system is necessary to ensure that permanent sequence numbers are observed later, e.g., in repair.

Optimistic Accept and Update—Q/U and FaB Paxos w/ Tentative Execution.

Two of the protocols that we survey use both optimistic accept and optimistic update phases. Q/U [1] is a Byzantine-fault-tolerant state-machine-replication protocol based on opaque quorum systems. FaB Paxos, which normally employs only an optimistic accept phase as described above, can, like the BFT variant described above, also employ an optimistic update optimization known as tentative execution. Since both Q/U and FaB Paxos with tentative execution always skip phase 2a (i.e., regardless of whether the execution is gracious), neither protocol can use fewer than $5b + 1$ replicas. In addition, since they also skip phase 3a, both protocols require the client to wait for a quorum of identical responses in phase 4 to make certain that the result is based on a permanent sequence number assignment; this quorum is larger than quorums in systems that employ dissemination or masking quorum systems.

Q/U. Figures 2.10 and 2.11 show the Q/U protocol. In phase 1, clients act as proposers and directly issue requests to the replicas. Since there are multiple proposers, a proposer may not know the next sequence number (implemented as a logical timestamp). Therefore, the client first retrieves the update history (called a replica history set) from a quorum of replicas (phase 1a). A quorum of replica history sets is called an

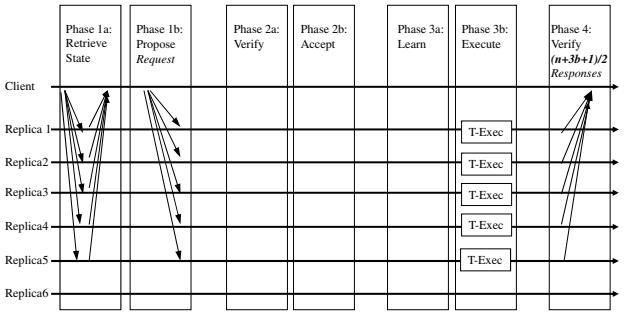


Figure 2.10: The Q/U protocol (optimistic accept and update).

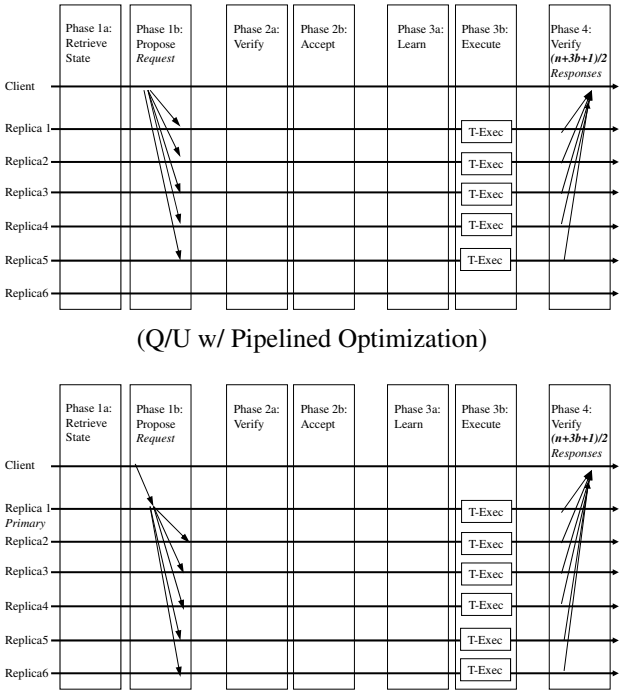


Figure 2.11: Optimistic accept and update in Q/U and FaB Paxos (both optimized).

object history set. It identifies the latest completed update, and, therefore, the sequence number at which the next update should be applied. The client sends the object history set along with the request to a quorum of replicas (phase 1b). In phase 2b, each replica verifies that it has not executed any operation more recent than that which is reflected in the object history set, and then accepts the update. Having accepted the update, the acceptor executes the request (phase 3b). Because Q/U is an optimistic execution protocol (it skips phase 3a), the client must wait for a quorum of responses in phase 4.

In a pipelined optimization of Q/U (shown in Figure 2.11), clients cache object history sets after each operation. As such, clients can avoid phase 1a if no other clients have since updated the object.

FaB Paxos w/ Tentative Execution. The tentative-execution optimization for FaB Paxos works as it does in BFT—a replica executes the request upon accepting the sequence number assignment for it in phase 2b (assuming it has also executed the requests corresponding to all earlier permanent sequence numbers). Because this sequence number assignment may never become permanent, it may need to be rolled back. Therefore, clients must wait for a quorum of identical responses in phase 4. Figure 2.11 highlights the similarities between Q/U with the pipelined optimization described above and FaB Paxos with the tentative-execution optimization.

2.4.4 Other Trade-Offs

BFT, Zyzzyva, and FaB Paxos use a single proposer (the primary), and so can omit phase 1a. On the other hand, Q/U allows clients to act as proposers, and therefore requires phase 1a (though it can be avoided in some cases with the pipelined optimization). The use of a single proposer has potential advantages. First, a client sends only a single request to the system (in the common case) as opposed to sending the request to an entire quorum. Therefore, since the single proposer is likely physically closer than the clients to the replicas, the use of a primary might be more efficient, e.g., on a WAN with relatively large message delays. Another advantage is that request-batching optimizations can be employed because the primary is aware of requests from multiple clients. Furthermore, use of a primary can mitigate the impact of client contention. However, the use of a primary: (i) involves an extra message delay (for the request to be forwarded from the client to the primary); and (ii) may allow a faulty primary to slow progress without being detected.

Because a proposer may be Byzantine-faulty, a repair phase may be necessary in order for the service to make progress in the presence of faults.[2] In systems such as BFT and FaB Paxos that use a dedicated proposer, the repair phase is used in part to choose a new proposer. In Q/U, this phase may also result from concurrent client updates, and is used to make sure that non-faulty replicas no longer have conflicting sequence number assignments. In BFT and FaB Paxos, repair is initiated by non-faulty replicas that have learned of some request but not have executed it after a specified length of time (*proactive repair*). In Q/U, repair is initiated by a client that has learned that the system is in a state from which no update can be completed due to conflicting proposed sequence number assignments (*need-based repair*). Because it is based on timeouts, proactive repair might sometimes be executed when it is not actually of help, e.g., when the network is being slow but the primary is not faulty.

2.5 Summary

Differences in intersection properties (1.2), (1.3), and (1.4), as discussed below, require the various types of Byzantine quorum systems (dissemination, masking, and opaque) to make different assumptions. In particular, as seen in Table 2.1, they anticipate that different types of servers may accept conflicting writes. This has implications for the number of servers required, as well as for the number of rounds of communication needed to ensure a successful update.

In state-machine-replication protocols, an optimistic accept phase requires an opaque quorum system construction, and therefore at least $5b + 1$ servers; a pessimistic accept phase can use dissemination or masking quorum systems, and therefore as few as $3b + 1$ servers, but requires at least one additional round

[2] We do not classify protocols based on their repair phases. Therefore, we do not distinguish between BFT and SINTRA [17], for example.

of communication. An optimistic update phase allows replicas to process requests that may later be given a different ordering; a pessimistic update phase requires an additional round of communication, but allows replicas to be confident that the request ordering is permanent before processing.

Dissemination quorums. Property (1.2) of dissemination quorum systems requires that any two quorums overlap in at least one non-faulty replica. Since a quorum may contain more faulty replicas than this (up to b), dissemination quorum systems are suitable for *self-verifying* data. If the fault model does not allow for Byzantine-faulty clients, then clients can use digital signatures to make data self-verifying. Otherwise, data can be made self-verifying through multiple-rounds of signed (or, alternatively, authenticated [22]) messages as described in Section 2.3.2 and Section 2.4.2.

Masking quorums. Masking quorum systems make relatively simple assumptions. As seen in (1.3), faulty replicas can fabricate conflicting writes, but non-faulty replicas will not accept conflicting writes. This holds even for potentially "outdated" non-faulty replicas, i.e., those in Q' but not Q when Q was used to perform a write to the system. In PASIS-RW, timestamps are uniquely determined by the request, and therefore writes do not conflict. However, writes are not self-verifying, so a masking quorum system is used to prevent faulty servers from generating writes. If sequential timestamps must instead be assigned to requests as in state-machine-replication protocols (and therefore cannot be based on the requests themselves), one option for keeping potentially outdated replicas from accepting conflicting writes is to use an echo protocol like that described in Section 2.3.2 .

Opaque quorums. Opaque quorum systems can allow for write operations to complete in a single (optimistic) round of communication, even in the face of Byzantine and/or concurrent clients that may, e.g., propose conflicting writes to some non-faulty replicas. Their appropriateness stems from (1.4), which anticipates that any server in Q' but not Q may be outdated and therefore may accept a conflicting write.

Table 2.1: Threshold quorum systems.

	Example	conflicting write
opaque	Q/U [1], FaB Paxos [48]	any server
masking	PASIS [29]	faulty server
dissemination	BFT [22], Zyzzyva [35]	no server

Chapter 3

System Model and Definitions

Our results in the remainder of the dissertation are based on certain assumptions. Together, these assumptions form our system model, which we describe here. We also use this chapter to define terms that may not be familiar to the reader.

3.1 Conventions

Throughout the dissertation, we use San Serif to denote random variables, uppercase *ITALICS* for set-valued constants, and lowercase *italics* for integer-valued constants. In the literature, *random variables* are sometimes restricted only to functions that output real numbers; for clarity in distinguishing between either fixed sets or permutations and those that are sampled from a probability distribution, we use the term random variable to refer also to a function on a sample space that outputs either sets or permutations.

3.2 Fault Model

We assume a system with a set U of servers, $|U| = n$, and an arbitrary but bounded number of clients. Clients and servers can fail arbitrarily (i.e., Byzantine faults [37]). We assume that up to b servers can fail, and denote the set of faulty servers by B, where $B \subseteq U$. Any number of clients can fail. Failures are permanent. Clients and servers that do not fail are said to be *non-faulty*. We assume that faulty clients and servers all know the membership of B (although non-faulty clients and servers do not). However, as is typical for many Byzantine-fault-tolerant protocols (c.f., [1, 22, 29, 48]), we assume that faulty clients and servers are computationally bounded such that they cannot subvert standard cryptographic primitives such as digital signatures. The nature of the client puzzles used in the access-restriction protocol presented in Chapter 8 requires that each client must work in isolation; this can be achieved by restricting each user to single client via a secure ID assignment mechanism such as that used in [21].

3.3 Service Semantics and Consistency Model

We describe client operations as either *writes* that alter the state of the service or *reads* that do not. To help distinguish between writes, each write yields a corresponding *candidate* at some number of servers. A

candidate is an abstraction used in part to ensure that two distinct write operations are distinguishable from each other, even if the corresponding data values are the same.

The consistency model we seek is linearizability [31], also known as atomicity. Under linearizability, the service should appear as though it were non-replicated. Informally, a write that succeeds is called "established" (we define established more precisely in Section 3.4). Following the consistency model, a non-faulty client performs a read to obtain the candidate of the latest established write, where "latest" refers most recent write preceding this read in a linearization of the execution. We define the *correct* candidate for the read to return to be the candidate of this latest established write; other candidates are called *incorrect*.

If two writes are concurrent (e.g., they both bear the same timestamp, or are "conditioned on" the same established write in the sense used in Q/U [1]), at most one should become established. As such, no non-faulty server accepts both an established candidate and another that conflicts with it. Any candidate that is characterized by the property that a non-faulty server would accept either it or a given established candidate, but not both, is called a *conflicting* candidate. In either masking or opaque quorum systems, a faulty server may try to forge a conflicting candidate. In a dissemination quorum system, faulty servers are unable to forge conflicting candidates because the candidates are self-verifying [43].

3.4 Behavior of Benign Clients

We assume that the read and write operations by non-faulty clients take the following forms:

- **Writes:** To perform a write, a non-faulty client selects a *write access set* $A_{wt} \subseteq U$ of size a_{wt} uniformly at random and attempts to inform all servers in A_{wt} of its candidate. Formally, the write is *established* once all non-faulty servers in some set $Q_{wt} \subseteq A_{wt}$ of size $q_{wt} \leq a_{wt}$ servers have *accepted* this write. We refer to q_{wt} as the *write quorum size*; to any $Q_{wt} \subseteq U$ of that size as a *write quorum*; and to $\mathcal{Q}_{\mathrm{wt}} = \{Q_{wt} \subseteq U : |Q_{wt}| = q_{wt}\}$ as the *write quorum system*.

- **Reads:** To perform a read, a non-faulty client selects a *read access set* A_{rd} of size a_{rd} uniformly at random and attempts to contact each server in A_{rd} to learn the candidate that the server last accepted. We denote the minimum number of servers from which a non-faulty client must receive a response to complete the read successfully by $q_{rd} \leq a_{rd}$. We refer to q_{rd} as the *read quorum size*; to any $Q_{rd} \subseteq U$ of that size as a *read quorum*; and to $\mathcal{Q}_{\mathrm{rd}} = \{Q_{rd} \subseteq U : |Q_{rd}| = q_{rd}\}$ as the *read quorum system*.

3.5 Voting

In a read operation, we refer to each response received from a server in A_{rd} as a *vote* for a candidate. The read operation discerns the correct candidate from these votes in a protocol-specific way. A server can try to *vote* for some candidate (e.g., by responding to a read operation) if the server is a *participant* in voting (i.e., if the server is a member of the client's read access set). However for the technique of write markers presented in Chapter 7, we draw the distinction that a server becomes *qualified* to vote for a particular candidate only if the server is a member of the client's write access set selected for the write operation for which it votes.

3.6 Repair

Because the service is replicated, some candidates may not be (or ever become) established. These writes may have failed, or may simply be concurrent with the read. Therefore, it is possible that the (at least q_{rd}) votes received by a client may reflect a write operation but not provide enough evidence to determine whether that write is established. In this case, the reader may itself establish, or *repair*, the write value before returning it, to ensure that a subsequent reader returns that value as well (which is necessary to achieve our consistency model, linearizability). In such a protocol, the reader does so by copying its votes for that value to servers, in order to convince them to accept that write. Faulty clients can also repair values, as discussed below.

3.7 Access Strategy and Error

Previous work has treated access strategies as probability distributions on quorums. We strictly generalize the notion of access strategy to apply instead to *access sets* from which quorums are chosen. An access set is a set of servers from which the client selects a quorum. Intuitively, the analysis of access sets in addition to quorums is useful because a client can contact all servers in the access set in order to find a live quorum, c.f., [14].

As seen in Section 3.4, we adopt the access strategy that all access sets are chosen uniformly at random (even by faulty clients). Moreover, as described in Section 3.9, if the client is non-faulty, we assume that it chooses every read quorum from the corresponding read access set uniformly at random.

Informally, by following the access strategy, clients ensure that the service is consistent with high probability. Consistency means that the service satisfies properties P1 and P2 presented in Section 4.1. We are only concerned with reads by faulty clients when they can be used for repair. Consequently, an *error* is said to occur in a read operation when a non-faulty client fails to observe the latest candidate or a faulty client obtains sufficiently many votes for a conflicting candidate. This definition (or specifically "sufficiently many") will be made more precise in Section 6.2.

The *error probability* is the probability of an error when the client (non-faulty or faulty) reads from a read access set A_{rd} chosen uniformly at random. While we cannot force a faulty client to choose A_{rd} uniformly at random, in Chapter 8 we demonstrate an access protocol that enables a faulty client to assemble votes for a value that can be verified by servers only if A_{rd} was selected uniformly at random, which is good enough for our purposes. So, from here forward, we restrict our attention to read access sets chosen in this way. Our analysis allows that access sets may be larger than quorums, though if access sets and quorums are of the same size, then our protocol effectively forces even faulty clients to select quorums uniformly at random.

3.8 Behavior of Faulty Clients

We assume that faulty clients seek to maximize the error probability by following specific strategies. This is a conservative assumption; a client cannot increase—but may decrease—the probability of error by failing to follow these strategies. We defer the discussion of these strategies to Chapter 4, where we have additional context for their explanation.

3.9 Communication Assumptions

The communication assumptions we adopt are common to prior works in probabilistic [47] and signed [72] quorum systems: we assume that each non-faulty client can successfully communicate with each non-faulty server with high probability, and hence with all non-faulty servers with roughly equal probability. This assumption is in place to ensure that the network does not significantly bias a non-faulty client's interactions with servers either toward faulty servers or toward different non-faulty servers than those with which another non-faulty client can interact. Put another way, we treat a server that can be reliably reached by none or only some non-faulty clients as a member of B.

This assumption enables us to refine the read protocol of Section 3.4 in a straightforward way so that non-faulty clients choose read quorums from an access set uniformly at random. (More precisely, a faulty server can bias quorum selection away from quorums containing it by not responding, but this decreases the error probability, and so we conservatively assume that non-faulty clients select read quorums at random from their access sets.) However, because a write is, by definition, established once all of the non-faulty servers in any write quorum within A_{wt} have accepted it, the write quorum at which a write is established contains all servers in $A_{wt} \cap B$; i.e., only the the non-faulty servers within the write quorum are selected uniformly at random by a non-faulty client.

The access-restriction protocol of Chapter 8 requires no communication assumptions beyond those of the probabilistic quorums it supports. Similarly, write markers require no communication assumptions beyond those of the probabilistic quorums for which they are used.

Chapter 4

Basics of Probabilistic Quorum Systems

In this chapter, we review the concept of probabilistic quorum systems in detail. We begin with a discussion of the strategies of faulty clients. We then present updated constraints for probabilistic dissemination and masking quorum systems in the context of our access strategy.

4.1 Consistency Properties

Probabilistic quorum systems must satisfy constraints similar to those of strict quorum systems (e.g., (1.2), (1.3), or (1.4)), but only with probability $1 - \epsilon$. Given that the stated assumptions of a strict opaque quorum system hold, the system behaves correctly. In contrast to this, probabilistic quorum systems allow for a (small) possibility of error. Informally, this can be thought of as relaxing the relevant consistency constraint so that a variant of it holds for most—but not all—quorums. As with strict quorum systems, the purpose of these constraints is to guarantee that operations can be observed consistently in subsequent operations by receiving enough votes:

P1 Every read observes sufficiently many votes for the correct value to identify it as such.

P2 No (non-faulty or faulty) reader obtains votes for a conflicting value sufficient to repair it successfully.

To ensure that the probability of an error happening is small, probabilistic quorum systems are designed so that P1 and P2 hold with high probability. P1 ensures that a read by a non-faulty client always returns the correct value. (Note that the presence of a correct value implies the existence of at most one established candidate.) P2 ensures that no faulty client is able to gather enough votes for a conflicting value as a result of a read to repair that value.

4.2 Behavior of Faulty Clients

By using the access restriction protocol of Chapter 8 we can constrain faulty clients so that they must choose access sets uniformly at random. However, from each access set, a faulty client can choose the quorum so as to maximize the probability that an error happens. Here, we examine the way that a faulty client should choose quorums to maximize the chance of error.

Throughout this section, let A_{wt} denote a write access set from which Q_{wt} (a quorum used for an established write) is selected by a faulty client, let A'_{wt} be a write access set used for a conflicting write by a faulty client, and let A_{rd} be a read access set from which Q_{rd}, a read quorum, is selected by a faulty client. Again, we assume that A_{wt}, A'_{wt}, and A_{rd} are selected uniformly at random, an assumption that can be enforced using the protocol of Chapter 8.

4.2.1 Reads by a Faulty Client

During a read, a faulty client seeks a quorum that violates P2 to use for repair. For this reason, the correctness requirements discussed in Section 4.3 treat the number of votes that a faulty client can gather for a conflicting candidate (i.e., one that should not be repaired successfully) instead of only the number of votes that a non-faulty reader observes for the correct value. Enabling a faulty client to obtain sufficiently many votes for a conflicting value would, e.g., enable it to convince non-faulty servers to accept the conflicting value via the repair protocol, a possibility that must be avoided for correctness.

A faulty client can increase the probability that P2 is violated by choosing a read quorum with the most faulty servers and non-faulty servers that share the same conflicting value. Because a faulty client can collude with the servers in B, it can obtain replies from all servers in B that are also in A_{rd}, i.e., the servers in $A_{rd} \cap B$. It can also wait for responses from all of the non-faulty servers in A_{rd} with the conflicting value, i.e., those in $A_{rd} \cap (A'_{wt} \setminus Q_{wt})$. Only after receiving all such responses, and only if these responses number fewer than q_{rd}, must it choose responses from servers with other values.

4.2.2 Writes by a Faulty Client

During a write, a faulty client seeks a write quorum that: (i) maximizes the probability that P1 is violated on a subsequent read by a non-faulty client; or (ii) maximizes the probability that P2 is violated on a subsequent read by any client (even itself or another client that is faulty).

Therefore, a faulty client: (i) when establishing a candidate, writes the candidate to as few non-faulty servers as possible to minimize the probability that it is observed by a non-faulty client; and (ii) writes a conflicting candidate to as many servers as will accept it (i.e., faulty servers plus, in the case of an opaque quorum system, any non-faulty server that has not accepted the established candidate, i.e., $A'_{wt} \setminus Q_{wt}$) in order to maximize the probability that it is observed. Since a faulty client may perform *both* such writes, we assume that this client has knowledge of A_{wt} and A'_{wt} simultaneously. However, it is important to note that a faulty client does not have knowledge of the read access set A'_{rd} used by a non-faulty client—or specifically the non-faulty servers within it, i.e., $A'_{rd} \setminus B$—and so Q_{wt} is chosen independently of $A'_{rd} \setminus B$.[1]

Goal (i) requires maximizing $|Q_{wt} \cap B|$ to maximize the probability that P1 is violated; hence, first preference is given to the servers in $A_{wt} \cap B$ in a write. Goal (ii) requires minimizing $|(Q_{wt} \cap A'_{wt}) \setminus B|$ to maximize the probability that P2 is violated; hence, the servers in $(A_{wt} \cap A'_{wt}) \setminus B$ are avoided to the extent possible.

[1] More precisely, with the access protocol in Chapter 8, A'_{rd} can be hidden unless, and until, that read access set is used for repair, at which point it is too late for faulty clients to choose Q_{wt} so as to induce an error in that read operation.

4.3 Consistency Constraints

We now review the constraints of probabilistic dissemination and masking quorum systems. The constraints must preserve P1 and P2. First, the constraints must ensure in expectation that a non-faulty client can observe the latest established candidate if such a candidate exists. Let Q_{rd} represent a read quorum chosen uniformly at random, i.e., a random variable, from a read access set itself chosen uniformly at random. (Think of this quorum as one used by a non-faulty client.) Let Q_{wt} represent a write quorum chosen by a potentially faulty client; Q_{wt} must be chosen from A_{wt}, an access set chosen uniformly at random. (Think of Q_{wt} as a quorum used for an established candidate.) Then the threshold r number of votes necessary to observe a value must be less than the expected number of non-faulty qualified participants, which is

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right]. \tag{4.1}$$

Second, the constraints must ensure that a conflicting candidate (which is in conflict with an established candidate as described in Chapter 3) is, in expectation, not observed by any client (non-faulty or faulty). In general, it is important for all clients to observe only established candidates so as to enable higher-level protocols (e.g., [1]) that employ repair phases that may affect the state of the system within a read [53].

Probabilistic Dissemination Quorums. No server can return a conflicting candidate in a dissemination quorum system. Therefore, probabilistic dissemination quorum systems require a probabilistic version of (1.2), i.e.:

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > 0. \tag{4.2}$$

On the other hand, both masking and opaque quorum systems allow that some servers may return conflicting candidates. Let $\mathsf{A}'_{\mathrm{rd}}$ and $\mathsf{A}'_{\mathrm{wt}}$ represent read and write access sets, respectively, chosen uniformly at random. (Think of $\mathsf{A}'_{\mathrm{wt}}$ as the access set used by a faulty client for a conflicting candidate, and of $\mathsf{A}'_{\mathrm{rd}}$ as the access set used by a faulty client for a read operation. How faulty clients can be forced to choose uniformly at random is described in Chapter 8.)

Probabilistic Masking Quorums. In a masking quorum system, (1.3) dictates that only faulty servers may vote for a conflicting candidate. In the context of strict threshold quorums systems where b is the threshold number of faults, (1.3) might just as well be written

$$|(Q \cap Q') \setminus B| > b$$

for all $Q, Q' \in \mathcal{Q}$. Following this definition, one might be tempted to write an equivalent probabilistic constraint such as $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > b$. However, as identified in [47], this is sub-optimum in a probabilistic setting. The basic reason is that, in a probabilistic quorum system, $\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right]$ is less than b when a_{rd} is less than n as shown in (5.2). Thus, the following constraint is actually sufficient:

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right]. \tag{4.3}$$

4.4 Summary

Probabilistic quorum systems satisfy constraints similar to those of strict quorum systems, but only with probability $1 - \epsilon$. The probability of error is based on the model of client behavior as well as on the access strategy. The actual error probability for a deployed system might be lower in cases such as when there are fewer than b faulty servers or no faulty clients.

Chapter 5

Framework for Proofs

In this and subsequent chapters, we prove that various probabilistic quorum system constructions work so long as certain constraints are met. Each proof is based on the framework presented in this chapter. Lemmas 5.2.3 and 5.2.4, together present basic requirements for the types of quorum systems with which we are concerned. To demonstrate the capability of the framework, we present new bounds for probabilistic masking quorum systems.

5.1 McDiarmid Concentration Bound

The following theorem is a restatement of the Molloy and Reed statement [57, p. 172] of the McDiarmid Inequality that can be used to show that a random variable computed on a series of independent permutations is concentrated about its expectation.

Theorem 5.1.1 ([57])**.** *Let $\mathsf{Z} = \boldsymbol{z}(\Pi_1, \ldots, \Pi_l)$ be a random variable that is a non-negative function of a series $\Pi_1, \ldots, \Pi_l$ of independent random variables, where each Π_i takes on a random permutation (bijection) $\pi : \{1, \ldots, |P|\} \rightarrow P$ of a finite non-empty set P. Also, for some positive constants δ and μ, let the following conditions hold (where if $\Pi_j = \pi_j$ then the* mapping *$\langle i, j, m\rangle$ indicates that $\pi_j(i) = m$):*

- *M1 Swapping the mappings of any two elements in a single permutation π_j (i.e., changing $\{\langle i, j, m\rangle, \langle i', j, m'\rangle\}$ to $\{\langle i', j, m\rangle, \langle i, j, m'\rangle\}$, where $i \neq i'$ and $m \neq m'$) changes the value of Z by at most δ.*
- *M2 If $\mathsf{Z} = \boldsymbol{z}(\pi_1, \ldots, \pi_l) = x$, then there exists a set of at most μx distinct mappings $\{\langle i_1, j_1, m_1\rangle, \ldots, \langle i_{\mu x}, j_{\mu x}, m_{\mu x}\rangle\}$ such that $\boldsymbol{z}(\pi'_1, \ldots, \pi'_l) \geq x$ for any $\pi'_1, \ldots, \pi'_l$ sharing the same set of mappings.*

If $0 \leq \Delta \leq \mathbb{E}[\mathsf{Z}]$, then:

$$\Pr(|\mathsf{Z} - \mathbb{E}[\mathsf{Z}]| \geq \Delta + 60\delta\sqrt{\mu\mathbb{E}[\mathsf{Z}] + 1}) \leq 4/e^{\left(\Delta^2/8\delta^2\mu\mathbb{E}[\mathsf{Z}]\right)}.$$

We simplify Theorem 5.1.1 to create Corollary 5.1.2 that deals with asymptotic bounds.

Corollary 5.1.2. *Let $\mathsf{Z} = \boldsymbol{z}(\Pi_1, \ldots, \Pi_l)$ be a random variable that is a non-negative function of a series $\Pi_1, \ldots, \Pi_l$ of independent random variables, where each Π_i takes on a random permutation (bijection) $\pi : \{1, \ldots, |P|\} \rightarrow P$ of a finite non-empty set P. Also, for some positive constants δ and μ, let the following conditions hold (where if $\Pi_j = \pi_j$ then the* mapping *$\langle i, j, m\rangle$ indicates that $\pi_j(i) = m$):*

M1 *Swapping the mappings of any two elements in a single permutation π_j (i.e., changing $\{\langle i,j,m\rangle,\langle i',j,m'\rangle\}$ to $\{\langle i',j,m\rangle,\langle i,j,m'\rangle\}$, where $i \neq i'$ and $m \neq m'$) changes the value of Z by at most δ.*

M2 *If $\mathsf{Z} = \boldsymbol{z}(\pi_1,\ldots,\pi_l) = x$, then there exists a set of at most μx distinct mappings $\{\langle i_1,j_1,m_1\rangle,\ldots,\langle i_{\mu x},j_{\mu x},m_{\mu x}\rangle\}$ such that $\boldsymbol{z}(\pi'_1,\ldots,\pi'_l) \geq x$ for any $\pi'_1,\ldots,\pi'_l$ sharing the same set of mappings.*

If $\Delta = \omega(\sqrt{\mathbb{E}[\mathsf{Z}]})$, then,[1]

$$\Pr(|\mathsf{Z} - \mathbb{E}[\mathsf{Z}]| \geq \Delta) = 2/e^{(\omega(1))} \quad \textit{as } \Delta \to \infty.$$

Proof of Corollary 5.1.2. Since $\Delta = \omega(\sqrt{\mathbb{E}[\mathsf{Z}]})$, the $60\delta\sqrt{\mu\mathbb{E}[\mathsf{Z}]} + 1$ term is negligible, and, for any constant $\beta < 1/8\delta^2\mu$ and large enough value of $\mathbb{E}[\mathsf{Z}]$, we have (c.f., [57, p. 81]),

$$\Pr(|\mathsf{Z} - \mathbb{E}[\mathsf{Z}]| \geq \Delta) \leq 2/e^{\left(\beta\Delta^2/\mathbb{E}[\mathsf{Z}]\right)}.$$

In other words, if $\Delta = \omega(\sqrt{\mathbb{E}[\mathsf{Z}]})$, then,

$$\Pr(|\mathsf{Z} - \mathbb{E}[\mathsf{Z}]| \geq \Delta) = 2/e^{(\omega(1))} \quad \text{as } \Delta \to \infty. \qquad \square$$

5.2 General Bound for Probabilistic Quorums

So that we can apply Corollary 5.1.2 to bound the error probability, we present a method for defining Q_{wt}, $\mathsf{A}'_{\mathrm{rd}}$, A_{wt}, $\mathsf{A}'_{\mathrm{wt}}$, and Q_{rd} (where Q_{rd} represents a read quorum selected by a non-faulty client) in terms of independent random variables Π_1, Π_2, Π_3, and Π_4, each taking on a random permutation $\{1,\ldots,|U|\} \to U$, where U is the set of all n servers. Fix any set of b servers to constitute B. Then consider the following definitions:

- Define $\mathsf{A}_{\mathrm{wt}} = \{\Pi_1(1),\ldots,\Pi_1(a_{wt})\}$.
- Define $\mathsf{A}'_{\mathrm{wt}} = \{\Pi_2(1),\ldots,\Pi_2(a_{wt})\}$.
- Define $\mathsf{A}'_{\mathrm{rd}} = \{\Pi_3(1),\ldots,\Pi_3(a_{rd})\}$.
- Define $\mathsf{Q}_{\mathrm{rd}} = \{\Pi_4(1),\ldots,\Pi_4(q_{rd})\}$.

Because each permutation is randomly selected (independently of B), so too are A_{wt}, $\mathsf{A}'_{\mathrm{wt}}$, $\mathsf{A}'_{\mathrm{rd}}$, and Q_{rd}. We define Q_{wt} in accordance with Chapter 3. Specifically, first we choose each $\Pi_1(i)$ such that $\Pi_1(i) \in \mathsf{A}_{\mathrm{wt}} \cap B$. Next, for each $j = 1..a_{wt}$, we choose $\Pi_1(j)$ if we have not yet chosen q_{wt} servers and $\Pi_1(j) \in \mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)$. Finally, for each $k = 1..a_{wt}$, we choose $\Pi_1(k)$ if we have not yet chosen q_{wt} servers and $\Pi_1(k) \in \mathsf{A}_{\mathrm{wt}} \cap (\mathsf{A}'_{\mathrm{wt}} \setminus B)$.

Let r be the threshold, discussed in Chapter 4, for the number of votes necessary to observe a candidate. Define $\mathsf{MinCorrect}$ to be a random variable for the number of non-faulty servers with the established candidate, i.e., $\mathsf{MinCorrect} = |(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|$ as indicated in (4.1).

[1] ω is the little-oh analog of Ω, i.e., $f(n) = \omega(g(n))$ if $f(n)/g(n) \to \infty$ as $n \to \infty$.

Lemma 5.2.1. *Let* $\mathsf{Z} = \mathsf{MinCorrect}$. *Let* $\mathbb{E}[\mathsf{Z}] > r$ *and* $\mathbb{E}[\mathsf{Z}] - r = \omega(\sqrt{\mathbb{E}[\mathsf{Z}]})$. *Then,*

$$\Pr(\mathsf{Z} \leq r) = 2/e^{(\omega(1))} \quad \textit{as } \mathbb{E}[\mathsf{Z}] - r \to \infty.$$

Proof of Lemma 5.2.1. For a fixed B, MinCorrect can be treated as a function of independent permutations (i.e., using Q_{rd} and Q_{wt}, as defined above). Consider this in relation to Corollary 5.1.2 as follows. Swapping any two elements in either Q_{rd} or Q_{wt} can change the value of MinCorrect by at most 1; therefore, $\delta = 1$ in Condition M1. Additionally, if $\mathsf{MinCorrect} = x$, then the mappings

$$\bigcup_{u \in (\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B} \{\langle \Pi_1^{-1}(u), 1, u\rangle, \langle \Pi_4^{-1}(u), 4, u\rangle\}$$

suffice to satisfy Condition M2; therefore, $\mu = 2$.

Let $\Delta = \mathbb{E}[\mathsf{Z}] - r$; then, by assumption, $\Delta = \omega(\sqrt{\mathbb{E}[\mathsf{Z}]})$. We apply Corollary 5.1.2, yielding,

$$\begin{aligned}
&\Pr(\mathsf{Z} \leq r) \\
&= \Pr(\mathsf{Z} \leq \mathbb{E}[\mathsf{Z}] - \Delta) \\
&= \Pr(\mathbb{E}[\mathsf{Z}] - \mathsf{Z} \geq \Delta) \\
&\leq \Pr(|\mathsf{Z} - \mathbb{E}[\mathsf{Z}]| \geq \Delta) \\
&= 2/e^{(\omega(1))} \quad \text{as } \Delta \to \infty \\
&= 2/e^{(\omega(1))} \quad \text{as } \mathbb{E}[\mathsf{Z}] - r \to \infty.
\end{aligned}$$

□

Define MaxConflicting to be a random variable for the maximum number of servers that vote for a conflicting candidate. For example: due to (4.2), in dissemination quorums, MaxConflicting takes on the value 0 with probability 1; due to (4.3), in masking quorums without write markers, $\mathsf{MaxConflicting} = |\mathsf{A}'_{\mathrm{rd}} \cap B|$; due to (6.2), in opaque quorum systems without write markers, $\mathsf{MaxConflicting} = |\mathsf{A}'_{\mathrm{rd}} \cap B| + |((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B) \setminus \mathsf{Q}_{\mathrm{wt}}|$; due to (7.1), in masking quorums with write markers, $\mathsf{MaxConflicting} = |(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|$; and due to (7.2), in opaque quorums with write markers, $\mathsf{MaxConflicting} = |(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B| + |((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B) \setminus \mathsf{Q}_{\mathrm{wt}}|$.

Lemma 5.2.2. *Let* $\mathsf{Z}' = \mathsf{MaxConflicting}$. *Let* $r > \mathbb{E}[\mathsf{Z}']$ *and* $r - \mathbb{E}[\mathsf{Z}'] = \omega(\sqrt{\mathbb{E}[\mathsf{Z}']})$. *Then,*

$$\Pr(\mathsf{Z}' \geq r) = 2/e^{(\omega(1))} \quad \textit{as } r - \mathbb{E}[\mathsf{Z}'] \to \infty.$$

Proof of Lemma 5.2.2. As seen in Chapters 4, 6, and 7, depending on the type of quorum system, MaxConflicting has a different definition. However, for each type of quorum system, for a fixed B, MaxConflicting can be treated as a function of independent permutations from the set $\{\mathsf{A}'_{\mathrm{rd}}, \mathsf{A}'_{\mathrm{wt}}, \mathsf{Q}_{\mathrm{wt}}\}$. Consider this in relation to Corollary 5.1.2 as follows. Swapping any two elements in one permutation can change the value of MaxConflicting by at most 1 because an additional server added to $\mathsf{A}'_{\mathrm{rd}}$ cannot be both faulty and non-faulty; therefore, $\delta = 1$ in Condition M1. Additionally, for any value of MaxConflicting considered in Chapters 4, 6, and 7, if $\mathsf{MaxConflicting} = |C| = x$, then the mappings

$$\bigcup_{u \in C} \{\langle \Pi_1^{-1}(u), 1, u\rangle, \langle \Pi_2^{-1}(u), 2, u\rangle, \langle \Pi_3^{-1}(u), 3, u\rangle\}$$

suffice to satisfy Condition M2; therefore, $\mu \leq 3$.

Let $\Delta = r - \mathbb{E}[\mathsf{Z}']$; then, by assumption, $\Delta = \omega(\sqrt{\mathbb{E}[\mathsf{Z}']})$. We apply Corollary 5.1.2, yielding,

$$\begin{aligned}
&\Pr(\mathsf{Z}' \geq r) \\
&= \Pr(\mathsf{Z}' \geq \Delta + \mathbb{E}\left[\mathsf{Z}'\right]) \\
&= \Pr(\mathsf{Z}' - \mathbb{E}\left[\mathsf{Z}'\right] \geq \Delta) \\
&\leq \Pr(|\mathsf{Z}' - \mathbb{E}\left[\mathsf{Z}'\right]| \geq \Delta) \\
&= 2/e^{(\omega(1))} \quad \text{as } \Delta \to \infty \\
&= 2/e^{(\omega(1))} \quad \text{as } r - \mathbb{E}[\mathsf{Z}'] \to \infty.
\end{aligned}$$

□

Lemma 5.2.3. *Let $n - b = \Omega(n)$. For all $c > 0$ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$ and $q_{rd}q_{wt} - n = \Omega(1)$, it is the case that $\mathbb{E}[\mathsf{MinCorrect}] > c$ for all n sufficiently large.*

Proof of Lemma 5.2.3. Note that,

$$\begin{aligned}
\mathbb{E}[\mathsf{MinCorrect}] &= q_{rd}q_{wt}\left(\frac{n-b}{n^2}\right) \\
&> dn\left(\frac{n-b}{n^2}\right) \\
&= d\left(\frac{n-b}{n}\right) \\
&= d\left(\frac{\Omega(n)}{n}\right) \\
&= d\left(\Omega(1)\right).
\end{aligned}$$

□

Lemma 5.2.4. *Let the following hold,*

$$\begin{aligned}
&\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}] > 0, \\
&\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}] = \omega(\sqrt{\mathbb{E}[\mathsf{MinCorrect}]}).
\end{aligned}$$

Then it is possible to set r such that,

$$\textit{error probability} \to 0 \quad \textit{as } \mathbb{E}[\mathsf{MinCorrect}] \to \infty.$$

Proof of Lemma 5.2.4. Set r as follows,

$$r = \frac{\mathbb{E}[\mathsf{MinCorrect}] + \mathbb{E}[\mathsf{MaxConflicting}]}{2}.$$

Then we can apply Lemma 5.2.1 to $\Pr(\mathsf{MinCorrect} \leq r)$ because,

$$\begin{aligned}
&\mathbb{E}[\mathsf{MinCorrect}] > r, \text{ and} \\
&\mathbb{E}[\mathsf{MinCorrect}] - r = \omega(\sqrt{\mathbb{E}[\mathsf{MinCorrect}]}).
\end{aligned}$$

Next, note that by assumption and our setting of r, $r - \mathbb{E}[\mathsf{MaxConflicting}] = \omega(\sqrt{\mathbb{E}[\mathsf{MinCorrect}]})$. But, since $\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}]$ grows when $\mathbb{E}[\mathsf{MinCorrect}]$ grows, it must be that $\mathbb{E}[\mathsf{MinCorrect}]$ grows faster than $\mathbb{E}[\mathsf{MaxConflicting}]$. Therefore, $r - \mathbb{E}[\mathsf{MaxConflicting}] = \omega(\sqrt{\mathbb{E}[\mathsf{MaxConflicting}]})$. As such, we can apply Lemma 5.2.2 to $\Pr(\mathsf{MaxConflicting} \geq r)$ because

$$r > \mathbb{E}[\mathsf{MaxConflicting}], \text{ and}$$
$$r - \mathbb{E}[\mathsf{MaxConflicting}] = \omega(\sqrt{\mathbb{E}[\mathsf{MaxConflicting}]}).$$

It is an error if $\mathsf{MinCorrect} < r$ or $\mathsf{MaxConflicting} \geq r$. Therefore, the error probability is bounded as follows:

$$\begin{aligned}
\text{error probability} &= \Pr(\mathsf{MaxConflicting} \geq r \vee \mathsf{MinCorrect} < r) \\
&= \Pr(\mathsf{MaxConflicting} \geq r) + \Pr(\mathsf{MinCorrect} < r) - \\
&\qquad \Pr(\mathsf{MaxConflicting} \geq r \wedge \mathsf{MinCorrect} < r) \\
&\leq \Pr(\mathsf{MaxConflicting} \geq r) + \Pr(\mathsf{MinCorrect} < r) \\
&\leq \Pr(\mathsf{MaxConflicting} \geq r) + \Pr(\mathsf{MinCorrect} \leq r) \\
&= 2/e^{(\omega(1))} + 2/e^{(\omega(1))} \\
&\qquad \text{as } (\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}])/2 \to \infty \\
&= 2/e^{(\omega(1))} + 2/e^{(\omega(1))} \quad \text{as } \mathbb{E}[\mathsf{MinCorrect}] \to \infty.
\end{aligned}$$

Where the second-to-last line follows because,

$$(\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}])/2 = \mathbb{E}[\mathsf{MinCorrect}] - r = r - \mathbb{E}[\mathsf{MaxConflicting}].$$

And the final line follows because,

$$(\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}])/2 = \omega(\sqrt{\mathbb{E}[\mathsf{MinCorrect}]}). \qquad \square$$

Here, a suitable setting of r is one between $\mathbb{E}[\mathsf{MinCorrect}]$ and $\mathbb{E}[\mathsf{MaxConflicting}]$, inclusive.

5.3 Benefits Over Chernoff Bounds

In our framework, we use the McDiarmid inequality rather than Chernoff bounds. Chernoff bounds are used in [47] to show that a probabilistic masking quorum system can tolerate $b < n/3$. With our proof framework, we show that a probabilistic masking quorum system can actually tolerate more faults, specifically $b < n/2.62$.

The intuition behind our improved result is that we can use (4.3) directly with our framework instead of only approximating it. For our purposes, Chernoff bounds are useful for bounding a random variable based on a single uniform sample of a population containing two types of elements. For example, Malkhi et al. [47] treat the servers as a population consisting of b faulty servers and $n - b$ non-faulty servers. Thus, they are able to use a Chernoff bound for the deviation from $\mathbb{E}[|\mathsf{A}'_{\mathrm{rd}} \cap B|]$ because A_{rd} represents a uniform sample. However, $|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|$ is based on two samples (i.e., Q_{rd} and Q_{wt}) from this population. Therefore, in order to use a Chernoff bound for the deviation from $\mathbb{E}[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|]$ they pessimistically assume that the number of faulty servers sampled in one of the samples is maximized. We need make no such assumption with our proof framework, as we demonstrate next.

5.4 Reevaluation: Probabilistic Dissemination and Masking Quorums

In this section, we are concerned with quorum systems for which we can achieve error probability (as defined in Chapter 3) no greater than a given ϵ for any n sufficiently large. For such quorum systems, there is an upper bound on b in terms of n, akin to the bound for strict quorum systems. Intuitively, the maximum value of b is limited by the relevant constraint (i.e., either (4.2), (4.3)). Of primary interest are Theorem 5.4.9 and its corollaries.

The remainder of the section is focused on determining, for each type of probabilistic quorum system, the upper bound on b and bounds on the load that Lemmas 5.2.3 and 5.2.4 imply. Important expected values, derived below using the worst-case behavior of faulty clients presented in Section 3.8, are as follows,

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] = \frac{q_{rd}(nq_{wt} - a_{wt}b)}{n^2}. \tag{5.1}$$

$$\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] = \frac{a_{rd}b}{n}. \tag{5.2}$$

Lemma 5.4.1.

$$\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] = \frac{a_{rd}b}{n}.$$

Proof. $\mathsf{A}'_{\mathrm{rd}}$ is selected independently of B. As such, $|\mathsf{A}'_{\mathrm{rd}} \cap B|$ is a hypergeometric random variable characterized by a_{rd} draws from a population of n elements containing b successes. Therefore, we use the formula for the expected value of a hypergeometric random variable. □

In the proofs of the following lemmas, we use rules of conditional expectation (c.f., [56, Section 2.3]). In particular, the following.

Definition 5.4.2 ([56])**.** *The expression* $\mathbb{E}[\mathsf{X} \mid \mathsf{Y}]$ *is a random variable* $f(\mathsf{Y})$ *that takes on the value* $\mathbb{E}[\mathsf{X} \mid \mathsf{Y} = y]$ *when* $\mathsf{Y} = y$*.*

Because $\mathbb{E}[\mathsf{X} \mid \mathsf{Y}]$ is a random variable, i.e., a function, it makes sense to consider its expectation.

Theorem 5.4.3 ([56, Theorem 2.7])**.**

$$\mathbb{E}[\mathsf{X}] = \mathbb{E}\left[\mathbb{E}[\mathsf{X} \mid \mathsf{Y}]\right]. \tag{5.3}$$

Lemma 5.4.4.

$$\mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right] = \frac{q_{wt}n - a_{wt}b}{n}. \tag{5.4}$$

Proof. Let $\mathsf{MalWrite} = |\mathsf{A}_{\mathrm{wt}} \cap B|$. Since A_{wt} is selected uniformly at random independently of B, $\mathsf{MalWrite}$ is a hypergeometric random variable, characterized by a_{wt} draws from a population of n elements containing b successes; therefore,

$$\mathbb{E}[\mathsf{MalWrite}] = \frac{a_{wt}b}{n}.$$

Recall from Chapter 3, that a write is established once all of the non-faulty servers in any write quorum in A_{wt} have accepted it. Therefore,

$$\mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B| \mid \mathsf{MalWrite} = m\right] = q_{wt} - m.$$

Applying Theorem 5.4.3 and linearity of expectation, we have that,

$$\begin{aligned}
&\mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right] \\
&\quad = \mathbb{E}\left[\mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B| \mid \mathsf{MalWrite}\right]\right] \\
&\quad = \mathbb{E}\left[q_{wt} - \mathsf{MalWrite}\right] \\
&\quad = q_{wt} - \mathbb{E}\left[\mathsf{MalWrite}\right] \\
&\quad = q_{wt} - \frac{a_{wt}b}{n}.
\end{aligned}$$

□

Lemma 5.4.5.

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] = \frac{q_{rd}(nq_{wt} - a_{wt}b)}{n^2}.$$

Proof. Q_{rd} is independent of $\mathsf{Q}_{\mathrm{wt}} \setminus B$; therefore, $|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B| \mid |\mathsf{Q}_{\mathrm{wt}} \setminus B| = m$ is a conditional hypergeometric random variable characterized by q_{rd} draws from a population of n elements containing m successes, and,

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B| \mid |\mathsf{Q}_{\mathrm{wt}} \setminus B| = m\right] = \frac{q_{rd}m}{n}.$$

Applying Theorem 5.4.3, by linearity of expectation and Lemma 5.4.4 we have that,

$$\begin{aligned}
&\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] \\
&= \mathbb{E}\left[\,\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B| \,|\, |\mathsf{Q}_{\mathrm{wt}} \setminus B|\right]\,\right] \\
&= \mathbb{E}\left[\frac{q_{rd}}{n}\left(|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right)\right] \\
&= \frac{q_{rd}}{n}\mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right] \\
&= \frac{q_{rd}(nq_{wt} - a_{wt}b)}{n^2}.
\end{aligned}$$

□

Lemma 5.4.6. *For a probabilistic dissemination or masking quorum system configuration without write markers,*

$$\begin{aligned}
&\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right] \Rightarrow \\
&\quad \mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]}).
\end{aligned}$$

Proof of Lemma 5.4.6. Consider:

- Dissemination without write markers: $\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]$ implies $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] - 0 \neq 0$, and so (5.1) show us that $\mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]})$.

- Masking without write markers: $\mathbb{E}[\mathsf{MinCorrect}] > \mathbb{E}[\mathsf{MaxConflicting}]$ implies $\mathbb{E}[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|] - \mathbb{E}[|\mathsf{A}'_{\mathrm{rd}} \cap B|] \neq 0$, and so (5.1) and (5.2) show us that $\mathbb{E}[\mathsf{MinCorrect}] - \mathbb{E}[\mathsf{MaxConflicting}] = \omega(\sqrt{\mathbb{E}[\mathsf{MinCorrect}]})$.

□

Theorem 5.4.7. *For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{wt}n}{a_{wt}},$$

any such probabilistic dissemination quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Proof of Theorem 5.4.7. We show that Lemmas 5.2.3 and 5.2.4 apply. By (5.1) and (4.2),

$$\begin{aligned} & b < \frac{q_{wt}n}{a_{wt}} \\ & \Leftrightarrow \mathbb{E}[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|] > 0 \\ & \Leftrightarrow \mathbb{E}[\mathsf{MinCorrect}] > \mathbb{E}[\mathsf{MaxConflicting}]. \end{aligned}$$

Therefore, we can apply Lemma 5.4.6 and, thus, Lemma 5.2.4. By the restriction on b and the conditions of the Theorem, we can also apply Lemma 5.2.3. □

The following corollary of Theorem 5.4.7 finds the same bound on b as obtained in [47], though with more flexibility on the choice of q_{rd} and q_{wt}. (Here, q_{rd} and q_{wt} need not both be a constant multiple of $\sqrt{n}$.)

Corollary 5.4.8. *Let $a_{wt} = q_{wt}$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < n,$$

any such probabilistic dissemination quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Theorem 5.4.9. *For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{rd}q_{wt}n}{q_{rd}a_{wt} + a_{rd}n},$$

any such probabilistic masking quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Proof of Theorem 5.4.9. We show that Lemmas 5.2.3 and 5.2.4 apply. By (5.1), (5.2), and (4.3),

$$\begin{aligned} b &< \frac{q_{rd}q_{wt}n}{q_{rd}a_{wt} + a_{rd}n} \\ &\Leftrightarrow \mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] \\ &\Leftrightarrow \mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]. \end{aligned}$$

Therefore, we can apply Lemma 5.4.6 and, thus, Lemma 5.2.4. By the restriction on b and the conditions of the Theorem, we can also apply Lemma 5.2.3. □

Corollary 5.4.10. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt}$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{wt}n}{q_{wt} + n},$$

any such probabilistic masking quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

In the following corollary, we prove that $b < n/2.62$ is a sufficient constraint, in contrast to the constraint of $b < n/3$ in [47].

Corollary 5.4.11. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt} = n - b$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < n/2.62,$$

any such probabilistic masking quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

5.5 Summary

We have presented a proof framework for probabilistic quorum systems. The framework allows us to prove that probabilistic quorum systems in general can meet specific load and fault tolerance targets. Instead of using Chernoff bounds, the framework is based on a McDiarmid inequality which yields improved results for the fault tolerance of probabilistic masking quorum systems.

Chapter 6

Probabilistic Opaque Quorum Systems

We present a new type of probabilistic quorum system—the probabilistic opaque quorum system. We derive a suitable constraint, and, using the proof framework of Chapter 5, we analyze this constraint in terms of maximum fault tolerance and minimum load. We then calculate error probabilities for a range of concrete system sizes.

6.1 Consistency Constraints

A probabilistic opaque quorum system must satisfy a constraint similar to that of strict opaque quorum systems (i.e., (1.4)), but only with probability $1 - \epsilon$. As in Section 4.3, let $\mathsf{A}'_{\mathrm{rd}}$ and $\mathsf{A}'_{\mathrm{wt}}$ represent read and write access sets, respectively, chosen uniformly at random. (Think of $\mathsf{A}'_{\mathrm{wt}}$ as the access set used by a faulty client for a conflicting candidate, and of $\mathsf{A}'_{\mathrm{rd}}$ as the access set used by a faulty client for a read operation. How faulty clients can be forced to choose uniformly at random is described in Chapter 8.)

Probabilistic Opaque Quorums. In an opaque quorum system, (1.4) dictates that all servers outside of Q as well as all faulty servers may vote for a conflicting candidate. In the context of strict threshold quorums systems, (1.4) might just as well be written

$$|(Q \cap Q') \setminus B| > |\overline{(Q \cap Q') \setminus B}| \tag{6.1}$$

for all $Q, Q' \in \mathcal{Q}$. Directly translating this to the probabilistic setting, i.e., as $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > n - \mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right]$ is not as beneficial as considering the classes of servers that can return a conflicting value individually.

First, as in probabilistic masking quorum systems (c.f., Section 4.3), $\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right]$ is not necessarily b, and so it should be written $\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right]$.

Second, in this case, it is of benefit to consider additional writes for a similarly subtle reason. It is not the case that all of the non-qualified non-faulty servers will return a conflicting candidate. No, only those that are actually part of a write will do so. Thus, if a conflicting candidate is written to $\mathsf{A}'_{\mathrm{wt}}$, then the only non-qualified non-faulty servers that return this conflicting candidate are those that are also in $\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}$. As such, the following constraint works:

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]. \tag{6.2}$$

6.2 Implied Bounds

In this section, we are concerned with probabilistic opaque quorum systems for which we can achieve error probability (as defined in Chapter 3) no greater than a given ϵ for any n sufficiently large. For such quorum systems, there is an upper bound on b in terms of n, akin to the bound for strict quorum systems.

Intuitively, the maximum value of b is limited by (6.2). Of primary interest are Theorem 5.4.9 and its corollaries. An important expected value, derived below using the worst-case behavior of faulty clients presented in Section 3.8, is as follows,

$$\mathbb{E}\left[\left|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}\right|\right] \leq \frac{a_{rd}}{n^3}(2a_{wt}n^2 - na_{wt}b - q_{wt}n^2 - a_{wt}^2 n + a_{wt}^2 b). \tag{6.3}$$

Lemma 6.2.1.

$$\mathbb{E}\left[\left|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right|\right] = a_{rd}\left(\frac{a_{wt}}{n} - \left(\frac{a_{wt}}{n}\right)\left(\frac{b}{n}\right)\right). \tag{6.4}$$

Proof. We calculate $\mathbb{E}\left[\left|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right|\right]$ directly as follows. Consider an indicator random variable Ind_u, such that $\mathsf{Ind}_u = 1$ if $u \in (\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B$, and $\mathsf{Ind}_u = 0$ otherwise. For each $u \in U \setminus B$, we have $\Pr[\mathsf{Ind}_u = 1] = \frac{a_{rd}a_{wt}}{n^2}$, since A'_{rd} and A'_{wt} are chosen independently. By linearity of expectation:

$$\begin{aligned}\mathbb{E}\left[\left|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right|\right] &= \sum_{u \in U \setminus B} \Pr(\mathsf{Ind}_u = 1) \\ &= (n-b)\left(\frac{a_{rd}a_{wt}}{n^2}\right) = a_{rd}\left(\frac{a_{wt}}{n} - \left(\frac{a_{wt}}{n}\right)\left(\frac{b}{n}\right)\right).\end{aligned}$$

□

Lemma 6.2.2.

$$\mathbb{E}\left[\left|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)\right|\right] \geq \frac{a_{rd}}{n}\left(q_{wt} - \frac{a_{wt}}{n}\left(b + \frac{(n-a_{wt})(n-b)}{n}\right)\right). \tag{6.5}$$

Proof. In calculating $\mathbb{E}\left[\left|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)\right|\right]$, because a faulty client may perform *both* a write that becomes established at Q_{wt} and another write at $\mathsf{A}'_{\mathrm{wt}}$ that conflicts with the first write, we cannot assume that Q_{wt} is selected independently of $\mathsf{A}'_{\mathrm{wt}}$. Let $\mathsf{CI} = \mathsf{A}'_{\mathrm{wt}} \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)$. Then, in particular, as described in Chapter 3, such a client seeks to maximize $\mathbb{E}\left[\mathsf{MaxConflicting}\right]$, and therefore minimizes $\mathbb{E}\left[|\mathsf{CI}|\right]$ by choosing the servers for $\mathsf{Q}_{\mathrm{wt}} \setminus B$ from $\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)$ first. Thus,

$$\begin{aligned}\mathbb{E}\left[|\mathsf{CI}|\right] &= \mathbf{max}(\ 0\ ,\ \mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right] - \mathbb{E}\left[\left|\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)\right|\ \right]) \\ &\geq \mathbb{E}\left[|\mathsf{Q}_{\mathrm{wt}} \setminus B|\right] - \mathbb{E}\left[\left|\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)\right|\right].\end{aligned} \tag{6.6}$$

We calculate $\mathbb{E}\left[|\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)|\right]$ directly as follows. For clarity, note that $\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B) = (\mathsf{A}_{\mathrm{wt}} \setminus \mathsf{A}'_{\mathrm{wt}}) \setminus B$. Consider an indicator random variable Ind_u, such that $\mathsf{Ind}_u = 1$ if $u \in (\mathsf{A}_{\mathrm{wt}} \setminus \mathsf{A}'_{\mathrm{wt}}) \setminus B$, and $\mathsf{Ind}_u = 0$ otherwise. For each $u \in U \setminus B$, we have $\Pr[\mathsf{Ind}_u = 1] = \frac{a_{wt}(n-a_{wt})}{n^2}$, since $\mathsf{A}'_{\mathrm{wt}}$ and A_{wt} are independent. By linearity of expectation:

$$\mathbb{E}\left[\left|\mathsf{A}_{\mathrm{wt}} \setminus (\mathsf{A}'_{\mathrm{wt}} \cup B)\right|\right] = \sum_{u \in U \setminus B} \Pr(\mathsf{Ind}_u = 1) = (n-b)\left(\frac{a_{wt}(n-a_{wt})}{n^2}\right) = \frac{a_{wt}}{n^2}(n-a_{wt})(n-b). \tag{6.7}$$

By (6.6), (5.4), and (6.7), we have that,

$$\mathbb{E}\left[|\mathsf{CI}|\right] \geq \left(q_{wt} - \frac{a_{wt}b}{n}\right) - \frac{a_{wt}}{n^2}(n-a_{wt})(n-b) = q_{wt} - \frac{a_{wt}}{n}\left(b + \frac{(n-a_{wt})(n-b)}{n}\right). \tag{6.8}$$

Since A_{rd} is independent of CI, we see that $|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)| \mid |\mathsf{CI}| = c'$ is a hypergeometric random variable characterized by a_{rd} draws from a population of n elements containing c' successes, and,

$$\mathbb{E}\left[|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)| \mid |\mathsf{CI}| = c'\right] = \frac{a_{rd}c'}{n}.$$

Applying Theorem 5.4.3, by linearity of expectation we have that,

$$\begin{aligned}
&\mathbb{E}\left[|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B))|\right] \\
&\quad = \mathbb{E}\left[\ \mathbb{E}\left[|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)| \mid |\mathsf{CI}|\right]\ \right] \\
&\quad = \mathbb{E}\left[\frac{a_{rd}}{n}|\mathsf{CI}|\right] = \frac{a_{rd}}{n}\mathbb{E}\left[|\mathsf{CI}|\right].
\end{aligned} \tag{6.9}$$

Therefore, by (6.8) and (6.9) we have that,

$$\mathbb{E}\left[|(\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B))|\right] \geq \frac{a_{rd}}{n}\left(q_{wt} - \frac{a_{wt}}{n}\left(b + \frac{(n-a_{wt})(n-b)}{n}\right)\right). \qquad \square$$

Lemma 6.2.3.

$$\mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right] \leq \frac{a_{rd}}{n^3}(2a_{wt}n^2 - na_{wt}b - q_{wt}n^2 - a_{wt}^2 n + a_{wt}^2 b).$$

Proof. To calculate $\mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]$, first note that:

$$\begin{aligned}
&\mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right] \\
&= \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B|\right] - \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap (\mathsf{Q}_{\mathrm{wt}} \setminus B)|\right].
\end{aligned} \tag{6.10}$$

Combining and simplifying equations (6.4), (6.5), and (6.10), we obtain (6.3). $\square$

Lemma 6.2.4. *For a probabilistic opaque quorum system configuration,*

$$\begin{aligned}
&\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right] \Rightarrow \\
&\quad \mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]}).
\end{aligned}$$

Proof of Lemma 6.2.4. Consider:

- Opaque without write markers: $\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]$ implies $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] - (\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]) \neq 0$, and so (5.1), (5.2), and (6.3) show us that $\mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]})$.

$\square$

Theorem 6.2.5. *For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{n(a_{rd}q_{wt}n - 2a_{rd}a_{wt}n + a_{wt}^2 a_{rd} + q_{rd}q_{wt}n)}{n^2 a_{rd} - a_{rd}a_{wt}n + a_{wt}^2 a_{rd} + q_{rd}a_{wt}n},$$

any such probabilistic opaque quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Proof of Theorem 6.2.5. We show that Lemmas 5.2.3 and 5.2.4 apply. By (5.1), (5.2), (6.3), and (6.2),

$$\begin{aligned} b &< \frac{(a_{rd}q_{wt}n - 2a_{rd}a_{wt}n + a_{wt}^2 a_{rd} + q_{rd}q_{wt}n)n}{n^2 a_{rd} - a_{rd}a_{wt}n + a_{wt}^2 a_{rd} + q_{rd}a_{wt}n} \\ &\Rightarrow \mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] + \mathbb{E}\left[\left|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}\right|\right] \\ &\Leftrightarrow \mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]. \end{aligned}$$

Therefore, we can apply Lemma 6.2.4 and, thus, Lemma 5.2.4. By the restriction on b and the conditions of the Theorem, we can also apply Lemma 5.2.3. □

Corollary 6.2.6. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt}$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{wt}^2 n}{q_{wt}^2 + n^2},$$

any such probabilistic opaque quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Corollary 6.2.7. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt} = n - b$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < n/3.15,$$

any such probabilistic opaque quorum system without write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

6.3 Spectrum of System Sizes

As shown in Section 6.2, a construction exhibits decreasing error probability in the limit with increasing n if PO-Consistency holds. Therefore, the remainder of this section is concerned with interpreting the inequality in Theorem 6.2.5. Our analysis (summarized in Table 6.1) shows that the best bounds are provided when: (i) both types of quorums are as large as possible (while still ensuring an available quorum), i.e., $q_{rd} = q_{wt} = n - b$; and (ii), given (i), that access sets are as small as possible.

We first consider scenarios in which a read or write can be completed with a single access set because $a_{rd} = q_{rd} + b$ ("single-phase reads") and $a_{wt} = q_{wt} + b$ ("single-phase writes"). Then, we derive the better bounds that can be achieved if we set $a_{rd} < q_{rd} + b$ or $a_{wt} < q_{wt} + b$. Finally, we consider scenarios pertaining only to clients that are benign.

Table 6.1: Lower bounds on n for various configurations.

$\mathbf{n} >$	n	$n-b$	$n-2b$	eq.	Sec. 4.4.x
3.15b	-	$a_{rd}\ q_{rd}\ a_{wt}\ q_{wt}$	-	(6.19)	4, 5
3.83b	a_{rd}	$q_{rd}\ a_{wt}\ q_{wt}$	-	(6.17)	3, 4
4.00b	a_{wt}	$a_{rd}\ q_{rd}\ q_{wt}$	-	(6.15)	2, 4, 5
4.08b	-	$a_{rd}\ a_{wt}\ q_{wt}$	q_{rd}	(6.18)	3
4.56b	$a_{rd}\ a_{wt}$	$q_{rd}\ q_{wt}$	-	(6.11)	1, 2, 3, 4
4.73b	a_{wt}	$a_{rd}\ q_{wt}$	q_{rd}	(6.12)	1, 3
5.49b	-	$a_{rd}\ q_{rd}\ a_{wt}$	q_{wt}	(6.16)	2
6.07b	a_{rd}	$q_{rd}\ a_{wt}$	q_{wt}	(6.13)	1, 2
6.19b	-	$a_{rd}\ a_{wt}$	$q_{rd}\ q_{wt}$	(6.14)	1

Single-Phase Reads and Writes.

Figure 6.1(a) plots the results of solving the inequality in Theorem 6.2.5 for n when $q_{wt} = a_{wt} - b$, $q_{rd} = a_{rd} - b$, and the sizes of access sets are varied between $n-b$ and n. We observe that the best bound is found when $a_{wt} = a_{rd} = n$ and $q_{wt} = q_{rd} = n - b$. In this case, we require,

$$c = \left(\frac{5+\sqrt{17}}{2}\right) \approx 4.561552813, \qquad n > c \cdot b. \tag{6.11}$$

We make the lower bound on n progressively worse by decreasing the sizes of access sets (and therefore quorums). If we set $a_{rd} = n - b$ and $a_{wt} = n$, we find,

$$c = \left(3+\sqrt{3}\right) \approx 4.732050808, \qquad n > c \cdot b. \tag{6.12}$$

By decreasing q_{wt} as a result of decreasing a_{wt}, we find that we soon fail to break the $n > 5b$ bound of strict opaque quorum systems [43]. If we set $a_{rd} = n$ and $a_{wt} = n - b$, we require,

$$c \approx 6.065103370, \qquad n > c \cdot b. \tag{6.13}$$

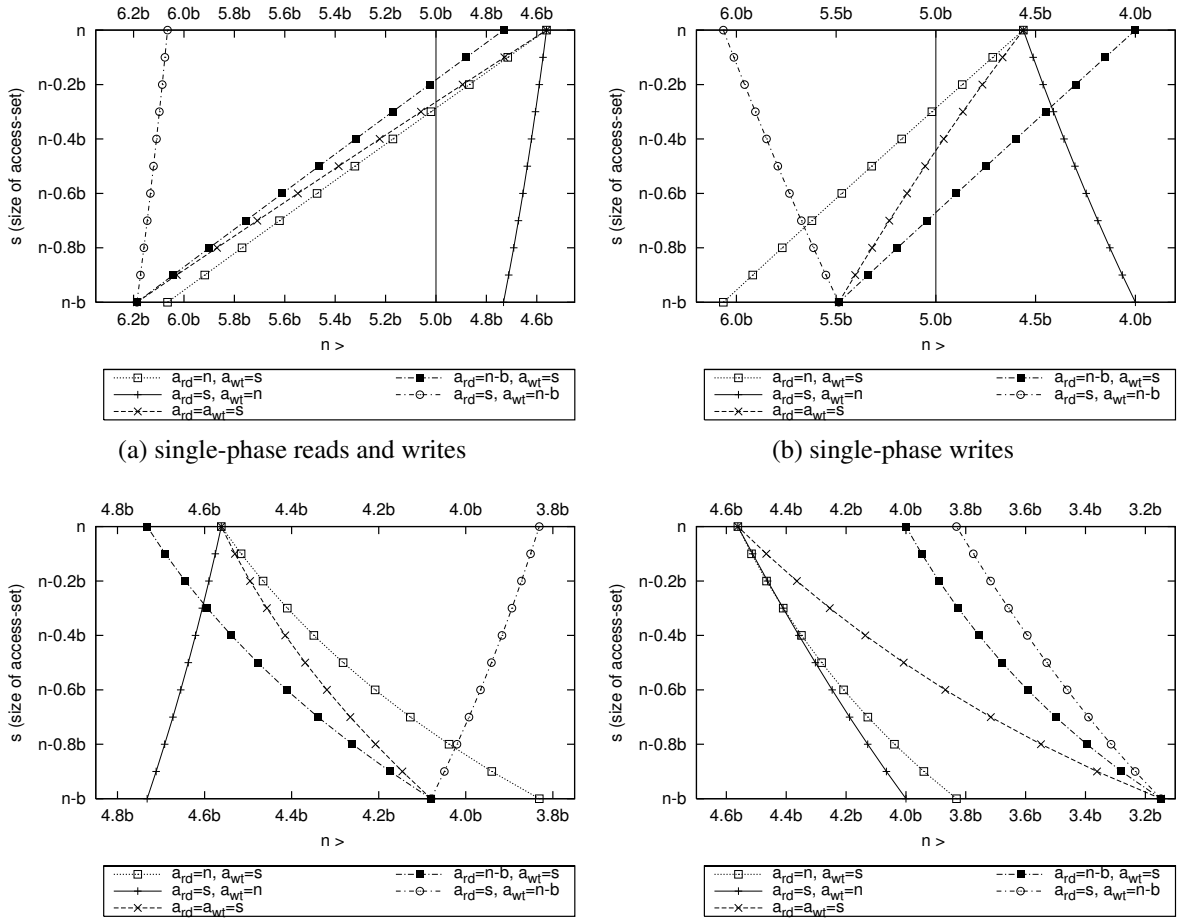
Finally, if we set $a_{rd} = a_{wt} = n - b$,

$$c \approx 6.186789391, \qquad n > c \cdot b. \tag{6.14}$$

Single-Phase Writes

Figure 6.1(b) plots the results of solving the inequality in Theorem 6.2.5 for n when $q_{wt} = a_{wt} - b$, $q_{rd} = n - b$, and the sizes of access sets are varied between $n - b$ and n. Because a read access set might not contain an available quorum, we can achieve better bounds than (6.11) by setting $a_{rd} < q_{rd} + b$. The best bound, when $a_{rd} = n - b$ and $a_{wt} = n$, is,

$$n > 4b. \tag{6.15}$$



(a) single-phase reads and writes

(b) single-phase writes

(c) single-phase reads

(d) best bounds

Figure 6.1: Sizes of access sets to achieve a given lower bound on n for: (a) $q_{rd} = a_{rd} - b$ and $q_{wt} = a_{wt} - b$; (b) $q_{rd} = n - b$ and $q_{wt} = a_{wt} - b$; (c) $q_{rd} = a_{rd} - b$ and $q_{wt} = n - b$; (d) $q_{rd} = q_{wt} = n - b$.

However, as in Section 6.3, decreasing q_{wt} by decreasing a_{wt} results in worse bounds, again quickly worse than $n > 5b$. If $a_{rd} = a_{wt} = n - b$ we require,

$$c \approx 5.486416764, \qquad n > c \cdot b. \tag{6.16}$$

The worst bound is when $a_{rd} = n$ and $a_{wt} = n - b$, i.e., (6.13).

Single-Phase Reads

Figure 6.1(c) plots the results of solving the inequality in Theorem 6.2.5 for n when $q_{wt} = n - b$, $q_{rd} = a_{rd} - b$, and the sizes of access sets are varied between $n - b$ and n. All of the points in the graph represent an improvement on the $n > 5b$ bound of strict opaque quorum systems. As in Section 6.3, we find that we can achieve better bounds than (6.11), here by decreasing the size of a_{wt}. For example, if $a_{wt} = n - b$ and

$a_{rd} = n$, we require,

$$c \approx 3.831177208, \qquad n > c \cdot b. \tag{6.17}$$

And if $a_{wt} = a_{rd} = n - b$, we require,

$$c \approx 4.079595625, \qquad n > c \cdot b. \tag{6.18}$$

The case where $a_{wt} = n$ and $a_{rd} = n - b$ is bound by (6.12).

Best Bounds

Now consider $q_{wt} = q_{rd} = n - b$. This results in better bounds than the scenarios in which the sizes of quorums are smaller. Figure 6.1(d) plots the results of solving the inequality in Theorem 6.2.5 for n when $q_{wt} = q_{rd} = n-b$, and the sizes of access sets are varied between $n-b$ and n. Because the sizes of quorums are fixed at $n - b$, we can improve on (6.11) by decreasing the size of a_{wt} or a_{rd}; the best bound, achieved when $a_{rd} = a_{wt} = n - b$, is,

$$c \approx 3.147899035, \qquad n > c \cdot b. \tag{6.19}$$

This represents the assumption in [47], in which all servers are accessed via randomly selected quorums. Otherwise, if $a_{rd} = n$ and $a_{wt} = n - b$ we have (6.17), and if $a_{rd} = n - b$ and $a_{wt} = n$, we have (6.15).

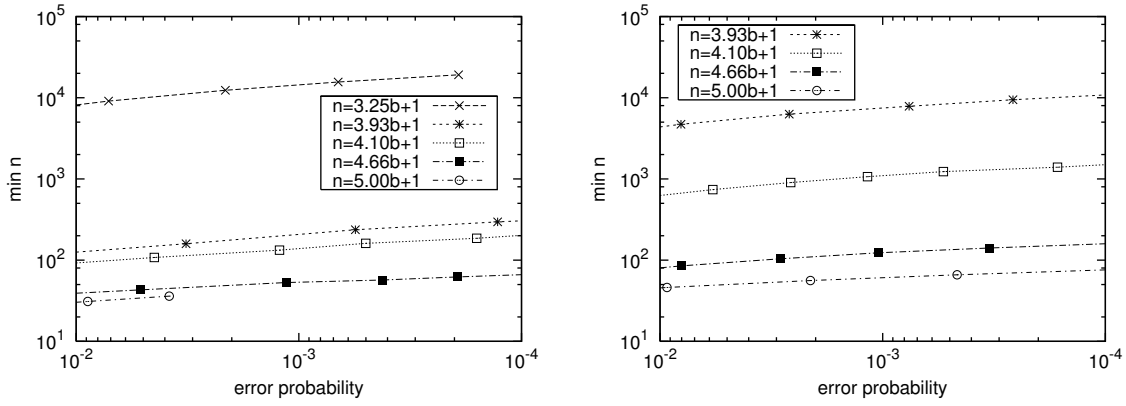
6.4 Error Probabilities

In this section, we analyze error probabilities for concrete system sizes. In addition to validating our above results, this shows that an access restriction protocol like that of Chapter 8 can provide significant advantages in terms of worst-case error probabilities.

Figure 6.2 plots the total number of nodes required to achieve a given calculated error probability for each of the configurations that tolerate faulty clients where $q_{wt} = q_{rd} = n - b$. Since the *unrestricted* configuration ($a_{rd} = n$, $a_{wt} = n$) shown in Figure 6.2(d) does not require the access-restriction protocol of Chapter 8, yet yields the best maximum ratios of b to n of all the configurations that provide single-phase reads and writes (Section 6.3), we do not evaluate the error probabilities of those other configurations here. In all cases, the error probabilities are worst-case in that they reflect the situation in which all b nodes are in fact faulty. For each configuration, we provide plots for different ratios of n to b, ranging from the maximum b for a given configuration, to $n = 5b + 1$, as a comparison with strict opaque quorum systems.

Overall, we find that our constructions can tolerate significantly more than $b = n/5$ faulty servers, while providing error probabilities in the range of 10^{-2} to 10^{-4} for systems with fewer than 50 servers to hundreds of servers. Coupled with the dissemination of correct values between servers (off the critical path), as described in Chapter 8, the error probability decreases between writes.

Within each figure, we see that to decrease the worst-case error probability, we can either keep the same function of b in terms of n while increasing n, or hold n fixed while decreasing the number of faults the system can tolerate. For example, Figure 6.2(b) shows that if $b = (n - 1)/4.66$, we decrease the worst-case error probability from $\sim 10^{-2}$ to $\sim 10^{-4}$ by increasing the system size from 48 servers to 141 servers. On the other hand, Figure 6.2(a) shows us that if we keep n fixed at ~ 100 servers, we can provide order of 10^{-3}



(a) restricted reads and writes ($a_{rd} = q_{rd}$, $a_{wt} = q_{wt}$)

(b) restricted writes only ($a_{rd} = n$, $a_{wt} = q_{wt}$)

(c) restricted reads only ($a_{rd} = q_{rd}$, $a_{wt} = n$)

(d) unrestricted ($a_{rd} = n$, $a_{wt} = n$)

Figure 6.2: Number of servers required to achieve given calculated worst-case error probability.

worst-case error probability with $(n-1)/4.10$ faulty servers, but provide only order of 10^{-2} worst-case error probability for $(n-1)/3.93$ faulty servers.

Considering two figures together, we see that configurations that tolerate a larger b also provide better error probabilities for a given b. For example, Figure 6.2(a) shows that by restricting reads and writes, a system of approximately 130 servers can tolerate $(n-1)/4.10$ faults with a worst-case error probability on the order of 10^{-3}. By comparison, Figure 6.2(b) shows that, if we restrict only writes, the same degree of fault tolerance and low error probability requires more than 1000 servers. As such, an access restriction protocol like that of Chapter 8 provides real benefits in terms of worst-case error probabilities.

While a very large number of servers is required for any configuration to tolerate its theoretical limit on b with small error probability, each configuration can tolerate close to its limit with far fewer servers. For example, Figure 6.2(a) shows that, if we restrict reads and writes, it requires more than 10,000 servers to tolerate $(n-1)/3.25$ faults with worst-case error probability on the order of 10^{-3}. However, by decreasing the fraction of faults that can be tolerated to $(n-1)/3.93$, we can achieve the same error probability with ~ 200 servers—$1/50$ of the servers. However, this is not a linear function; if we again reduce the fraction

of servers we can tolerate by a similar amount to $(n-1)/4.66$, we reduce the minimum n to approximately 50 servers—only by $1/4$.

6.5 Summary

We have presented probabilistic opaque quorum systems, a new type of probabilistic quorum system, in order to increase the fraction of faults that can be tolerated by an optimistic approach from fewer than $n/5$ to as many as $n/3.15$. A probabilistic opaque quorum system provides the same properties as strict opaque quorum systems, but is probabilistic in the sense that quorums are not guaranteed to overlap in the number of servers required to ensure safety. However, we have proven that this error probability can be made arbitrarily small (for a given ratio of b to n).

Chapter 7

Write Markers for Probabilistic Quorum Systems

In this chapter we present a variant of probabilistic quorum systems that uses *write markers* in order to limit the extent to which Byzantine-faulty servers act together. Our masking and opaque probabilistic quorum systems with write markers have asymptotically better load than the bounds proven for masking and opaque quorum systems without write markers. Moreover, our new masking and opaque probabilistic quorum systems can tolerate an additional 24% and 17% of faulty replicas, respectively, compared with probabilistic quorum systems without write markers.

7.1 The Technique

Intuitively, when a client submits a write, the candidate is associated with a write marker. We require that the following three properties are guaranteed by an implementation of write markers:

W1. Every candidate has a write marker that identifies the access set chosen for the write;

W2. A verifiable write marker implies that the access set was selected uniformly at random (i.e., according to the access strategy);

W3. Every non-faulty client and server can verify a write marker.

When considering a candidate, non-faulty clients and servers verify the candidate's write marker. Because of this verification, no non-faulty node will accept a vote for a candidate unless the issuing server is qualified to vote for the candidate. Since each write access set is chosen uniformly at random (W2), the faulty servers that can vote for a candidate, i.e., the faulty qualified servers, are therefore a random subset of the faulty servers.

Thus, write markers remove the advantage enjoyed by faulty servers in strict and traditional-probabilistic masking and opaque quorum systems,

Table 7.1: Ability of a server to vote for a given candidate: $\bullet$ (traditional quorums); $\star$ (write markers).

Type of server	**Vote**
Non-faulty qualified participant	$\bullet\ \star$
Faulty qualified participant	$\bullet\ \star$
Non-faulty non-qualified participant	
Faulty non-qualified participant	$\bullet$

where any faulty participant can vote for any candidate—and therefore can collude to have a conflicting, potentially fabricated candidate chosen instead of an established candidate. This aspect of write markers is summarized in Table 7.1, which shows the impact of write markers in terms of the abilities of faulty and non-faulty servers to vote for a given candidate.

7.2 Consistency Constraints with Write Markers

The use of write markers has no impact here on (4.1) because $(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B$ contains no faulty servers. However, write markers do enable us to set r smaller, as the following shows.

7.2.1 Masking with Write Markers

Using write markers for probabilistic masking quorum systems, we require that the faulty qualified participants alone cannot produce sufficient votes for a candidate to be observed in expectation. Taking (4.1) into consideration, we require:

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right]. \tag{7.1}$$

Contrast this with (1.3) and with the consistency requirement for traditional probabilistic masking quorum systems [47] (adapted to consider access sets), which requires that the faulty participants (qualified or not) cannot produce sufficient votes for a candidate to be observed in expectation. Intuitively, the intersection between access sets can be smaller with write markers because the right-hand side of (7.1) is less than the right-hand side of (4.3) if $a_{wt} < n$.

7.2.2 Opaque with Write Markers

With write markers, we have the benefit, described above for probabilistic masking quorums, in terms of the number of faulty participants that can vote for a candidate in expectation. However, as shown in (1.4), opaque quorum systems must additionally consider the maximum number of non-faulty qualified participants that vote for the same conflicting candidate in expectation. As such, instead of (7.1), we have:

$$\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]. \tag{7.2}$$

Contrast this with the consistency requirement for traditional probabilistic opaque quorums (6.2). Intuitively, the intersection between access sets can be smaller with write markers because the right-hand side of (7.2) is less than the right-hand side of (6.2) if $a_{wt} < n$.

7.3 Bounds

In this subsection, we are concerned with quorum systems for which we can achieve error probability (as defined in Chapter 3) no greater than a given ϵ for any n sufficiently large. For such quorum systems, there is an upper bound on b in terms of n, akin to the bound for strict quorum systems.

Intuitively, the maximum value of b is limited by the relevant constraint (i.e., either (7.1) or (7.2)). Of primary interest are Theorem 7.3.3 and its corollaries, which demonstrate the benefits of write markers

for probabilistic masking quorum systems, and Theorem 7.3.6 and its corollaries, which demonstrate the benefits of write markers for probabilistic opaque quorum systems.

The remainder of the section is focused on determining, for each type of probabilistic quorum system, the upper bound on b and bounds on the load that Lemmas 5.2.3 and 5.2.4 imply. Important expected values, derived below, are as follows,

$$\mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] = \frac{a_{rd}a_{wt}b}{n^2}. \tag{7.3}$$

$$\tag{7.4}$$

Lemma 7.3.1.

$$\mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] = \frac{a_{rd}a_{wt}b}{n^2}.$$

Proof. We calculate $\mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right]$ directly as follows. Consider an indicator random variable Ind_u, such that $\mathsf{Ind}_u = 1$ if $u \in (\mathsf{A}_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B$, and $\mathsf{Ind}_u = 0$ otherwise. For each $u \in B$, we have $\Pr[\mathsf{Ind}_u = 1] = \frac{a_{rd}a_{wt}}{n^2}$, since A_{rd} and A'_{wt} are chosen independently. By linearity of expectation:

$$\mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] = \sum_{u \in B} \Pr(\mathsf{Ind}_u = 1) = b\left(\frac{a_{rd}a_{wt}}{n^2}\right). \qquad \square$$

Lemma 7.3.2. *For a probabilistic masking or opaque quorum system configuration with write markers,*

$$\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right] \Rightarrow$$
$$\mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]}).$$

Proof of Lemma 7.3.2. Consider:

- Masking with write markers: $\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]$ implies $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] - \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] \neq 0$, and so (5.1) and (7.3) show us that $\mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]})$.

- Opaque with write markers: $\mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]$ implies $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] - (\mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]) \neq 0$, and so (5.1), (7.3), and (6.3) show us that $\mathbb{E}\left[\mathsf{MinCorrect}\right] - \mathbb{E}\left[\mathsf{MaxConflicting}\right] = \omega(\sqrt{\mathbb{E}\left[\mathsf{MinCorrect}\right]})$. $\square$

Theorem 7.3.3. *For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{rd}q_{wt}n}{q_{rd}a_{wt} + a_{rd}a_{wt}},$$

any such probabilistic masking quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Proof of Theorem 7.3.3. We show that Lemmas 5.2.3 and 5.2.4 apply. By (5.1), (7.3), and (7.1),

$$\begin{aligned} b &< \frac{q_{rd}q_{wt}n}{q_{rd}a_{wt} + a_{rd}a_{wt}} \\ &\Leftrightarrow \mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] \\ &\Leftrightarrow \mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]. \end{aligned}$$

Therefore, we can apply Lemma 7.3.2 and, thus, Lemma 5.2.4. By the restriction on b and the conditions of the Theorem, we can also apply Lemma 5.2.3. □

Corollary 7.3.4. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt}$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < n/2,$$

any such probabilistic masking quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

In other words, with write markers, the size of quorums does not impact the maximum fraction of faults that can be tolerated when quorums are selected uniformly at random (i.e., when $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt}$).

Corollary 7.3.5. *Let $a_{rd} = q_{rd}$, $a_{wt} = q_{wt}$, and $b < n/2$. For all ϵ there is a constant $\rho > 1$ such that if $q_{rd} = q_{wt} = \rho\sqrt{n}$, any such probabilistic masking quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large, and has load*

$$\rho\sqrt{n}/n = O(1/\sqrt{n}).$$

Theorem 7.3.6. *For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{n(a_{rd}a_{wt}^2 + a_{rd}q_{wt}n + q_{rd}q_{wt}n - 2a_{rd}a_{wt}n)}{a_{wt}(a_{rd}a_{wt} + q_{rd}n)},$$

any such probabilistic opaque quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Proof of Theorem 7.3.6. We show that Lemmas 5.2.3 and 5.2.4 apply. By (5.1), (7.3), (6.3), and (7.2),

$$\begin{aligned} b &< \frac{n(a_{rd}a_{wt}^2 + a_{rd}q_{wt}n + q_{rd}q_{wt}n - 2a_{rd}a_{wt}n)}{a_{wt}(a_{rd}a_{wt} + q_{rd}n)} \\ &\Rightarrow \mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right] > \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right] \\ &\Leftrightarrow \mathbb{E}\left[\mathsf{MinCorrect}\right] > \mathbb{E}\left[\mathsf{MaxConflicting}\right]. \end{aligned}$$

Therefore, we can apply Lemma 7.3.2 and, thus, Lemma 5.2.4. By the restriction on b and the conditions of the Theorem, we can also apply Lemma 5.2.3. □

Corollary 7.3.7. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt}$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < \frac{q_{wt}n}{q_{wt} + n},$$

any such probabilistic opaque quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

Comparing Corollary 7.3.7 with Corollary 7.3.4, we see that in the opaque quorum case q_{wt} cannot be set independently of b.

Corollary 7.3.8. *Let $a_{rd} = q_{rd}$, $a_{wt} = q_{wt}$, and $b < (q_{wt}n)/(q_{wt} + n)$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$ and $q_{rd}q_{wt} - n = \Omega(1)$, any such probabilistic opaque quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large, and has load*

$$\Omega(b/n).$$

Corollary 7.3.9. *Let $b = \Omega(\sqrt{n})$. For all ϵ there is a constant $d > 1$ such that for all a_{rd}, a_{wt}, q_{rd}, q_{wt} where $a_{rd} = a_{wt} = q_{rd} = q_{wt} = lb$ for a value l such that $c' \geq l > n/(n - b)$ for some constant c', $(lb)^2 > dn$ and $(lb)^2 - n = \Omega(1)$, any such probabilistic opaque quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large, and has load*

$$O(b/n).$$

Corollary 7.3.10. *Let $a_{rd} = q_{rd}$ and $a_{wt} = q_{wt} = n - b$. For all ϵ there is a constant $d > 1$ such that for all q_{rd}, q_{wt} where $q_{rd}q_{wt} > dn$, $q_{rd}q_{wt} - n = \Omega(1)$, and*

$$b < n/2.62,$$

any such probabilistic opaque quorum system employing write markers achieves error probability no greater than ϵ given a suitable setting of r for all n sufficiently large.

7.4 Summary

We have presented write markers, a way to improve the load of masking and opaque quorum systems asymptotically. Moreover, our new masking and opaque probabilistic quorum systems with write markers can tolerate an additional 24% and 17% of faulty replicas, respectively, compared with the proven bounds of probabilistic quorum systems without write markers. Write markers achieve this by limiting the extent to which Byzantine-faulty servers may cooperate to provide incorrect values to clients.

Chapter 8

Protocol to Enforce the Access Strategy

Byzantine clients are problematic for all probabilistic quorum systems because the combination of high fault tolerance and low probability of error that can be achieved is based on the assumption that clients follow the access strategy. Therefore, we present a protocol with which we constrain clients to using pseudo-randomly selected access sets (sets of servers contacted in order to find quorums, c.f., [14]) of a prescribed size.

In the limit, we can set the sizes of access sets to be the sizes of quorums, thereby dictating that all clients use pseudo-randomly selected quorums. However, as shown in previous chapters, the notion of restricted access sets allows us a range of options in trading off the low error probability and high fault tolerance of completely random quorum selection, for the guarantee of an available quorum (a quorum in which all servers respond) in every access set. Recall from Chapter 3 that the need for read access sets to be selected uniformly at random is motivated by repair. As such, protocols that do not involve repair may not require this access-restriction protocol for read operations.

The remainder of this chapter is structured as follows: Section 8.1 discusses factors that influence the design of the protocol; Section 8.2 discusses the source of randomness provided by the servers; Section 8.3 discusses the how the sequence of access sets is computed; Section 8.4 outlines the responsibilities of servers concerning verification of the validity of access sets; and Section 8.5 provides details of the background propagation of correct values.

8.1 Design Constraints

The operation of our protocol is motivated by the implicit need for each access set to be chosen independently not only of other access sets, but also of the state of the system (for example, the values held by each server, the identities of faulty servers, etc.). This is necessary to prevent a faulty client from taking advantage of knowledge of this state when choosing an access set in order to perform an operation that increases the probability of error, such as writing incorrect values to as many non-faulty servers as possible that do not have the correct value.

Our protocol must balance conflicting constraints. First, a client may be forced to discard a randomly chosen access set—and choose another—because a given access set (of size less than b servers more than a quorum) might not contain an available quorum. However, for efficiency, we want to avoid additional rounds of communication for each operation. This precludes, for example, a protocol in which the servers collectively choose an access set at random and assign it to the client for every operation. As such, a client

must be able to choose from multiple access sets without involving the servers for each. Yet, a faulty client should be prevented from discarding access sets in order to choose the one that has the highest probability of causing an error given the current system state. In addition, we should ensure that a faulty client does not benefit from waiting for the system state to change in order to use a previously chosen access set that becomes more advantageous as a result of the change.

In our protocol, the client obtains one or more random values, each called a Verifiable Random Value (VRV), with the participation of non-faulty servers. Each VRV determines a unique, verifiable, ordered sequence of random access sets that the client can use; the client has no control over the sequence. To deter a client from discarding earlier access sets in the sequence for potentially more favorable access sets later in the sequence, the protocol imposes an exponentially increasing cost (in terms of computation) for the ability to use later access sets. The cost is implemented as a *client puzzle* [33]. We couple this with a facility for the propagation of the correct value in the background so that any advantages for a faulty client in the current system state are reduced if the client chooses to delay performing the operation while it explores later access sets. Finally, to deter a client from waiting for the system state to change, we tie the validity of a VRV (and its sequence of access sets) to the state of the system so that as execution proceeds, any unused access sets become invalid.

8.2 Obtaining a VRV

In order to get an access set, the client first must obtain a VRV from the servers. Servers implement a metering policy, in which each server responds to a request for a VRV only after a delay. The delay varies, such that it increases exponentially with the rate at which the client has requested VRVs during some recent interval of time—i.e., a client that has not requested a VRV recently will receive a VRV with little or no delay, whereas a client that has recently requested many VRVs will receive a VRV after a (potentially significant) delay. To offload work from servers to clients (e.g., for scalability), the servers can make it relatively more expensive (in terms of time) to ask for and receive a new VRV than to compute a given number of access sets (potentially for multiple operations) from a single VRV, using the mechanisms described below.

The VRV is characterized by the following properties:

- It can be created only with the consent of non-faulty servers;
- Its validity is tied to the state of the system, in the sense that as the system state evolves (possibly merely through the passage of time), eventually the VRV is invalidated;
- While it is valid, any non-faulty server can verify its validity and so will accept it.

The VRV must be created with the consent of non-faulty servers because otherwise faulty servers might collude to issue multiple VRVs to a faulty client with no delay. Therefore, l, the number of servers required for the issuance of a VRV, must be at least $b + 1$. However, of the non-faulty servers in the system, only those among the (at least $l - b$) used to issue a VRV will impose additional delay before issuing an additional VRV. Therefore, to minimize the time to get an additional VRV, a faulty client avoids involving servers that have issued VRVs recently. This strategy maximizes the number of VRVs to which the non-faulty server contributing to the fewest VRVs has contributed. Thus, once k VRVs have been issued, all $n - b$ non-faulty servers have contributed to the issuance of at least $\lfloor k(l-b)/(n-b) \rfloor$ of these k. Since all non-faulty servers

have contributed to at least this many VRVs, and the delay is exponential in this number, the time $T(k)$ required for a client to obtain k VRVs is:

$$T(k) = \Omega\left(\exp\left\lfloor \frac{k(l-b)}{n-b} \right\rfloor\right)$$

In practice, $T(k)$ for a client decays during periods in which that client does not request additional VRVs, so that a client that does not request VRVs for a period can obtain one with small delay.

The validity of the VRV (and its sequence of access sets) is tied to the state of the system so that as execution proceeds, any unused access sets become invalid. To implement this, the replication protocol may provide some piece of data that varies with the state of the system—the *Object History Set* in Q/U [1] is an example of this—with which the servers can compute a VRV, but, in the absence of a suitable value from the protocol, the VRV can include a timestamp (assuming that the non-faulty servers have roughly synchronized clocks). The VRV consists of this value together with a digital signature created using a (l, n)-threshold signature scheme (e.g., [67]), i.e., so that any set of l servers can together create the signature, but smaller sets of servers cannot. The signature scheme must be *strongly unforgeable* [10], meaning that an adversary, given a VRV, is not able to find other valid VRVs. This is necessary because otherwise a faulty client would be able to generate variations of a valid VRV until finding one from which to select an access set that causes an error (see below).

8.3 Choosing an Access Set

As motivated above: (i) the VRV determines a sequence of valid access sets; and (ii) a client puzzle must make it exponentially harder to use later access sets in the sequence than earlier ones. In addition, it is desirable for our protocol to satisfy the following requirements:

- Each VRV must determine only a single valid sequence of access sets. This is to prevent a faulty client from choosing a preferred sequence.
- The puzzle solutions must be easy to verify, so that verification costs do not limit the scalability of the system in terms of the number of requests.
- There must be a solution to each puzzle. Otherwise a non-faulty client might be unable to use any access set.
- No server can know the solution to the puzzle beforehand due to the Byzantine fault model. Otherwise, a faulty client could avoid the exponential work by asking a faulty server for the solution.

In our protocol, the sequence of access sets is determined as follows. Let v be a VRV, let **g** be a hash function modeled as a random oracle [15], and let **access_set** be a deterministic operation that, given a seed value, selects an access set of the specified size from the set of all access sets of that size in a uniform fashion. Let the first seed, s_1, be $\mathbf{g}(v)$, and the i'th seed, s_i, be $\mathbf{g}(s_{i-1})$. Then the i'th access set is **access_set**(s_i).

The function **access_set** works as follows for seed value s. Let **h** be a hash function modeled as a random oracle that maps inputs uniformly to its output range. Divide the output range of **h** into n equal sized intervals labeled 1 through n, and let **server**$(candidate)$ return the server corresponding to the range

into which $candidate$ falls. Let $candidate_1$ be $\mathbf{h}(s)$ and let $candidate_i$ be $\mathbf{h}(candidate_{i-1})$. Then the i'th candidate server chosen for the access set is $\mathbf{server}(candidate_i)$, and candidate servers are added to the access set until the access set contains the required number of unique servers. Because **h** maps inputs uniformly to its output range, each server is effectively selected at random.

In order to use the i'th access set, the client must solve a puzzle of suitable difficulty. This puzzle must be non-interactive [32] to avoid additional rounds of communication. There are many suitable candidate puzzle functions [32]; we use a simple variant of Hashcash [11]. Specifically, let **f** be a hash function modeled as a random oracle. Then, to use the i'th access-set, the client must find a value, w, such that the first ci bits of $\mathbf{f}(v||w)$ equal zero, where '||' represents concatenation and c is a constant greater than or equal to one. The most efficient known way to do so is a brute force search. In general, the client will need to compute $2^{ci}/2 = 2^{ci-1}$ evaluations of **f** to find a w such that the first ci bits of $\mathbf{f}(v||w)$ bits are zero. Larger values of c require more effort to obtain each access set.

8.4 Server Verification

Upon receiving a write request for the i'th access set, each non-faulty server in the chosen access set must verify that it is a member of the access set; for a repair request it must verify that the relevant votes are from servers in the access set of the read operation that gave rise to the repair. In addition, in either case, before accepting the value, each server must verify that the VRV is valid, that the access set corresponds to the i'th access set of the sequence, and that the client has provided a valid solution to a puzzle of difficulty ci. To validate the solution, w, the server need only check that the first ci bits of $\mathbf{f}(v||w)$ are zero.

While the client can obtain additional access sets from the VRV, each access set used is treated as a different operation by servers as stated in Section 3.4; e.g., a write operation using one access set, and then using another access set, is treated as two different writes,[1] so that a faulty client cannot "accumulate" more than a_{wt} servers for its operation through the use of multiple write access sets.

8.5 Background Propagation

As described above, servers work to propagate the values of established writes to each other in the background. We assume an appropriate propagation algorithm (e.g., a variant of an epidemic algorithm [24] such as [42]). At a high level, a non-faulty server has two responsibilities. First, having accepted a write value and returned a response to the client, it periodically informs other servers that it has accepted the value. Second, if it has not yet accepted a value upon learning that a threshold number, p, of servers have accepted the value, it accepts the value. Faulty servers are all assumed to have access to any conflicting value directly without propagation, so we assume no additional constraints on their behavior. In the following, we focus on the case of opaque quorum systems.

Probabilistic Opaque Quorums

Our main contribution is our analysis of the threshold number of servers that must propagate a value for it to be accepted by another server. While related Byzantine diffusion protocols (e.g., [42]) use the number

[1] Typically, a Byzantine-fault-tolerant write protocol must already be resilient to partial writes, which is how these writes using different access sets might appear to the service.

$b + 1$, we require a larger number because opaque quorum systems allow that some non-faulty servers may accept conflicting values.

Lemma 8.5.1. *Let $n < 2q_{wt} - 2b$ and $p = n - q_{wt} + b + 1$. Then an established value will be accepted and propagated by at least p non-faulty servers, and no conflicting value can be propagated by p servers (faulty or non-faulty).*

Proof. The established value is accepted by at least $q_{wt} - b$ non-faulty servers by definition. If we allow that all other servers may forward a conflicting value, then the number of servers that forward a conflicting value is $n - (q_{wt} - b)$. First, note that $p > n - (q_{wt} - b)$ by the lemma. In addition, $p \leq q_{wt} - b$ because,

$$\begin{aligned} & n < 2q_{wt} - 2b \\ & \Leftrightarrow n - (q_{wt} - b) < q_{wt} - b \\ & \Leftrightarrow n - (q_{wt} - b) + 1 \leq q_{wt} - b \\ & \Leftrightarrow p \leq q_{wt} - b \end{aligned}$$

□

For example, if $q_{wt} = n - b$ and $n > 4b$ we set $p = 2b + 1$. Since the established value will be accepted by at least p non-faulty servers, it will propagate. No conflicting value will propagate.

If the conditions of Lemma 8.5.1 do not hold, we must allow for some probability of error during propagation. We set p so that it is between the expectations of the minimum number of non-faulty servers that accept an established write (PCorrect), and the maximum number of servers that propagate a conflicting value (PConflicting). We can directly derive the expectation of PCorrect by observing from (6.2) that $\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right]$ is the expected value of a uniform sample of size q_{rd} from the servers counted by PCorrect. Therefore, by (5.1), we have:

$$\begin{aligned} \mathbb{E}\left[\mathsf{PCorrect}\right] &= \frac{n}{q_{rd}} \left(\mathbb{E}\left[|(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}) \setminus B|\right]\right) \\ &= \frac{n}{q_{rd}} \left(\frac{q_{rd}(nq_{wt} - a_{wt}b)}{n^2}\right) \\ &= \frac{nq_{wt} - a_{wt}b}{n}. \end{aligned} \tag{8.1}$$

In a similar fashion, based on (6.2), we know that $\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]$ is the expected number of servers in a uniform sample of size a_{rd} from the servers in PConflicting. Therefore, by (5.2) and (6.3), we have that:

$$\begin{aligned} \mathbb{E}\left[\mathsf{PConflicting}\right] &= \frac{n}{a_{rd}} \left(\mathbb{E}\left[|\mathsf{A}'_{\mathrm{rd}} \cap B|\right] + \mathbb{E}\left[|\left((\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}|\right]\right) \\ &= \frac{n}{a_{rd}} \left(\frac{a_{rd}b}{n} + \frac{a_{rd}}{n^3}(2a_{wt}n^2 - na_{wt}b - q_{wt}n^2 - a_{wt}^2 n + a_{wt}^2 b)\right) \\ &= \frac{n^2 b + 2a_{wt}n^2 - na_{wt}b - n^2 q_{wt} - a_{wt}^2 n + a_{wt}^2 b}{n^2}. \end{aligned} \tag{8.2}$$

Note that,

$$\begin{aligned}
&\mathbb{E}\left[|\mathsf{A}'_{\mathrm{wt}} \setminus (\mathsf{C}_{\mathrm{wt}} \cup B)|\right] \\
&= \mathbb{E}\left[|(\mathsf{A}'_{\mathrm{wt}} \setminus B) \setminus \mathsf{C}_{\mathrm{wt}}|\right] \\
&= \mathbb{E}\left[|\mathsf{A}'_{\mathrm{wt}} \setminus B| - |\mathsf{A}'_{\mathrm{wt}} \cap (\mathsf{C}_{\mathrm{wt}} \setminus B)|\right] \\
&= \mathbb{E}\left[|\mathsf{A}'_{\mathrm{wt}} \setminus B|\right] - \mathbb{E}\left[|\mathsf{A}'_{\mathrm{wt}} \cap \mathsf{C}_{\mathrm{wt}}|\right]. \qquad (8.3)
\end{aligned}$$

8.6 Summary

We have presented an optional, novel access-restriction protocol for probabilistic quorum systems that provides the ability for servers to constrain clients so that they use randomly selected access sets for operations.

Chapter 9

Implementation of Write Markers

We provide an implementation of write markers that achieves the behavior assumed in Chapter 7, even with Byzantine clients.

9.1 Adapting the Access-Restriction Protocol

Because clients may be faulty, we cannot rely on, e.g., digital signatures issued by them to implement write markers. Instead, we adapt mechanisms of our access-restriction protocol from Chapter 8. The access-restriction protocol is designed to ensure that all clients follow the access strategy. It already enables non-faulty *servers* to verify this before accepting a write. And, since it is the only way of which we are aware for a probabilistic quorum system to tolerate Byzantine clients when write markers are of benefit (i.e., when the sizes of write access sets are restricted), its mechanisms are appropriate.

The relevant parts of the preexisting protocol work as follows. From a pre-configured number of servers, a client obtains a *verifiable recent value* (VRV), the value of which is unpredictable to clients and b or fewer servers prior to its creation. This VRV is used to generate a pseudorandom sequence of access sets. Since a VRV can be verified using only public information, both it and the sequence of access sets it induces can be verified by clients and servers. Non-faulty clients choose the next unused access set for each operation. However, a faulty client is motivated to maximize the probability of error. If the use of the next access set in the sequence does not maximize the probability of error given the current state of the system (i.e., the candidates accepted by the servers), such a client may try to skip ahead some number of access sets. Alternatively, such a client might try to wait to use the next access set until the state of the system changes. If allowed to follow either strategy, such a client would circumvent the access strategy because its choice of access set would not be independent from the state of the system.

Three mechanisms are used together to coerce a faulty client to follow the access strategy. First, the client must perform exponentially increasing work in expectation in order to use later access sets. As such, a client requires exponentially increasing time in expectation in order to choose an access set. This is implemented by requiring that the client solve a client puzzle [33] of the appropriate difficulty. The solution to the puzzle is, in expectation, difficult to find but easy to verify. Second, the VRV and sequence of access sets become invalid as the non-faulty servers accept additional candidates, or as the system otherwise progresses (e.g., as time passes). Non-faulty servers verify that an access set is still valid, i.e., not stale, before accepting it. Thus, system progress forces the client to start its work anew, and, as such, makes the work solving

the puzzle for any unused access set wasted. Finally, during the time that the client is working, the established candidate propagates in the background to the non-faulty servers that are non-qualified (c.f., [42]). This decreases the window of vulnerability in which a given access set in the sequence is useful for a conflicting write by making non-qualified servers aware that (i) there is an established candidate (so that they will not accept a conflicting candidate) and (ii) that the state of the system has progressed (so that they will invalidate the current VRV if appropriate).

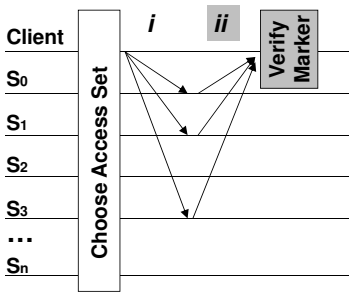


Figure 9.1: Read operation with write markers: messages and stages of verification of access set. (Changes in gray.)

The impact of these three mechanisms is that a non-faulty *server* can be confident that the choice of write access set adheres (at least approximately) to the access strategy upon having verified that the access set is valid, current, and is accompanied by an appropriate puzzle solution.

For write markers, we extend the protocol so that, as seen in Figure 9.1, *clients* can also perform verification. This requires that information about the puzzle solution and access set (including the VRV used to generate it) be returned by the servers to clients. (As seen in Figure 9.2 and explained below, this information varies across masking and opaque quorum systems.) In the preexisting access-restriction protocol, this information is verified and discarded by each server. For write markers, this information is instead stored by each server in the verification stage as a write marker. It is sent along with the data value as part of the candidate to the client during any read operation. If the server is non-faulty—a fact of which a non-faulty client cannot be certain—the access set used for the operation was indeed chosen according to the access strategy because the server performed verification before accepting the candidate. However, because the server may be faulty, the client performs verification as well; it verifies the write marker and that the server is a member of the access set. This allows us to guarantee points W1–W3 from Chapter 7. As such, faulty non-qualified servers are unable to vote for the candidates for which qualified servers can vote.

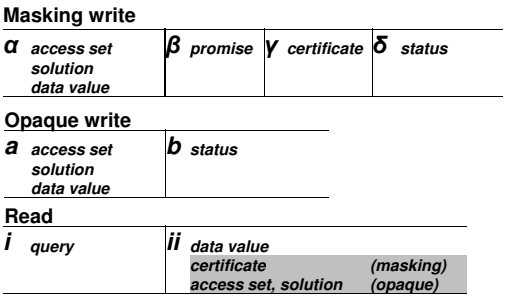


Figure 9.2: Message types. (Write marker emphasized with gray.)

Figures 9.1, 9.2, 9.3(a), and 9.3(b) illustrate relevant pieces of the preexisting protocol and our modifications for write markers in the context of read and write operations in probabilistic masking and opaque quorum systems. The figures highlight that the additions to the protocol for write markers involve saving the write markers and returning them to clients so that clients can also verify them.

The differences in the structure of the write marker for probabilistic opaque and masking quorum systems mentioned above results in subtly different guarantees. The remainder of the section discusses these details.

9.2 Implementation for Probabilistic Opaque Quorums

As seen in Figure 9.2 (message ii), a write marker for a probabilistic opaque quorum system consists of the write-access-set identifier (including the VRV) and the solution to the puzzle that unlocks the use of this access set. Unlike a non-faulty server that verifies the access set at the time of use, a non-faulty client cannot verify that an access set was not already stale when the access set was accepted by a faulty server. Initially, this may appear problematic because it is clear that, given sufficient time, a faulty client will eventually be able to solve the puzzle for its preferred access set to use for a conflicting write—this access set may contain all of the servers in B. In addition, the faulty client can delay the use of this access set because non-faulty clients will be unable to verify whether it was already stale when it was used.

Fortunately, because non-faulty servers will not accept a stale candidate (i.e., a candidate accompanied by a stale access set), the fact that a stale access set may be accepted by a faulty server does not impact the benefit of write markers for opaque quorum systems. In general, consistency requires (7.2), i.e.,

$$\mathbb{E}\left[\left|\left(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}\right) \setminus B\right|\right] > \mathbb{E}\left[\left|\left(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}\right) \cap B\right|\right] + \mathbb{E}\left[\left|\left(\left(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}\right) \setminus B\right) \setminus \mathsf{Q}_{\mathrm{wt}}\right|\right].$$

However, only faulty servers will accept a stale candidate. Therefore, if the candidate was stale when written to A'_{wt}, no non-faulty server would have accepted it. Thus, in this case, the consistency constraint is equivalent to,

$$\mathbb{E}\left[\left|\left(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}\right) \setminus B\right|\right] > \mathbb{E}\left[\left|\left(\mathsf{A}'_{\mathrm{rd}} \cap \mathsf{A}'_{\mathrm{wt}}\right) \cap B\right|\right].$$

Even if the access set contains all of the faulty servers, i.e., $B \subset A'_{wt}$, then this becomes,

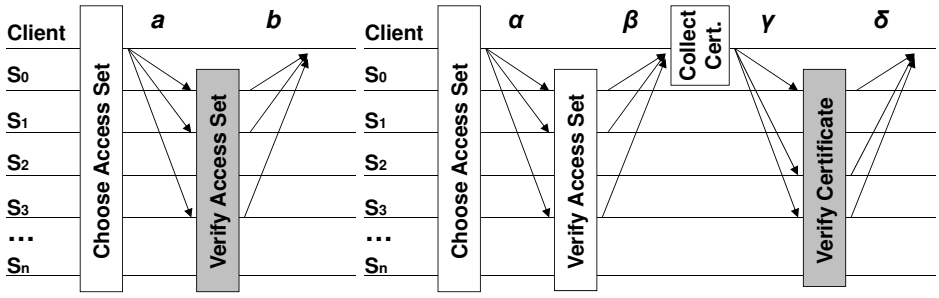
$$\mathbb{E}\left[\left|\left(\mathsf{Q}_{\mathrm{rd}} \cap \mathsf{Q}_{\mathrm{wt}}\right) \setminus B\right|\right] > \mathbb{E}\left[\left|\mathsf{A}'_{\mathrm{rd}} \cap B\right|\right].$$

However, this is (4.3), the constraint on probabilistic masking quorum systems without write markers. In effect, a faulty client must either: (i) use a recent access set that is therefore chosen approximately uniformly at random, and be limited by (7.2); or (ii), use a stale access set and be limited by (4.3). If quorums are the sizes of access sets, both inequalities have the same upper bound on b as seen in Corollary 5.4.10 and Corollary 7.3.7; otherwise, a faulty client is disadvantaged by using a stale access set because a system that satisfies (4.3) can tolerate more faults than one that satisfies (7.2), and is therefore less likely to result in error. (Compare the bounds in Theorem 5.4.9 and Theorem 7.3.6.)

9.3 Implementation for Probabilistic Masking Quorums

Protocols for masking quorum systems involve an additional round of communication (an echo phase, c.f., [44] or broadcast phase, c.f., [49]) during write operations in order to tolerate Byzantine or concurrent clients. This round prevents non-faulty servers from accepting conflicting data values, as assumed by (1.3). In order to write a data value, a client must first obtain a *write certificate* (a quorum of replies that together attest that the non-faulty servers will accept no conflicting data value). In contrast to optimistic protocols that use opaque quorum systems, these protocols are pessimistic.

This additional round allows us to prevent clients from using stale access sets. Specifically, in the request to authorize a data value (message α in Figure 9.2 and Figure 9.3(b)), the client sends the access set identifier (including the VRV), the solution to the puzzle enabling use of this access set, and the data value.



(a) Opaque quorums (b) Masking quorums

Figure 9.3: Write operation with write markers: messages and stages of verification. (Changes in gray.)

We require that the certificate come from servers in the access set that is chosen for the write operation. Each server verifies the VRV and that the puzzle solution enables use of the indicated access set before returning authorization (message β in Figure 9.2 and Figure 9.3(b)). The non-faulty servers that contribute to the certificate all implicitly agree that the access set is not stale, for otherwise they would not agree to the write. This certificate (sent to each server in message γ in Figure 9.2 and Figure 9.3(b)) is stored along with the data value as a write marker. Thus, unlike in probabilistic opaque quorum systems, a verifiable write marker in a probabilistic masking quorum system implies that a stale access set was not used. The reading client verifies the certificate (returned in message ii in Figure 9.1 and Figure 9.2) before accepting a vote for a candidate. Because a writing client will be unable to obtain a certificate for a stale access set, votes for such a candidate will be rejected by reading clients. Therefore, the analysis in Chapter 7 applies without additional complications.

9.4 Summary

We have presented a proposed implementation of write markers that is designed to be effective even while tolerating Byzantine-faulty clients and servers.

Chapter 10

The Cost of State Transfer

In many protocols, to switch quorums results in additional cost arising from the need to transfer *state* from servers in the old quorum to servers only in the new quorum. This is of particular concern on a wide area network where bandwidth is limited and delays are relatively large. Here, we discuss the following: (i) how much switching quorums impacts the performance of the Q/U protocol on a wide area network; and (ii) what the ideal access strategy is for a quorum-system protocol that involves state transfer when quorums are switched.

To make our discussion more concrete, we focus on state machine replication protocols [66], which, as discussed in Chapter 2, are an important class of protocol that require state transfer. In these protocols, the state of the service is replicated to different machines. The Q/U protocol [1] introduced the concept of a purely quorum-based state machine replication protocol, but protocols such as BFT [22], FaB Paxos [48], and Zyzzyva [35] also use quorum system properties to maintain state. Because updates to the state are generally incremental and based on the previous state, a replica typically needs a copy of the current state in order to perform such an operation. As this state can be modified at a quorum of servers, some servers may not always have the current state. Transferring the state to them is where the costs of state transfer become an issue.

Because Q/U uses an opaque quorum system, most servers are in every quorum and therefore have the current state. (Q/U quorums contain $4b + 1$ of $5b + 1$ total servers, where b is the number of faults that can be tolerated.) Even so, the Q/U authors have found the cost of state transfer to be relatively large even on a switched local area network where a peak throughput reduction of roughly 40% was observed [1] (see below). To avoid this cost, the authors introduce the notion of a *preferred quorum* [1]. In Q/U, the state is divided into *objects*. Each object is accessed at its preferred quorum by default. Since Q/U was originally evaluated on local area networks with homogeneous, low latencies, the preferred quorums are spread across the quorum system to balance load. The reduction in throughput mentioned above occurs when objects are accessed at quorums chosen uniformly at random—thereby requiring state-transfer—instead of at their preferred quorums.

10.1 Avoiding State Transfer on WANs

Two access strategies are known to perform well for certain workloads on wide area networks [60]. One, denoted ClosestDly, is to access the closest quorum in terms of network (but not processing) latency. In

other words, for each client c, $\mathsf{ClosestDly}^c$ assigns equal probability to each quorum that is closest to c, and $\mathsf{ClosestDly}^c(Q) = 0$ for all other quorums Q. ClosestDly works well when the workload is sufficiently light that the processing load imposed on servers contributes a negligible amount to the response times observed by clients. The second strategy, denoted Balanced, causes each client to access quorums uniformly at random, thereby balancing load on servers, assuming that each server is in the same number of quorums. (This is the case for all quorum systems we consider here.) That is, for each client c, $\mathsf{Balanced}^c$ assigns the same probability to each $Q \in \mathcal{Q}$. This access strategy makes sense when the workload is sufficiently heavy that response times are overwhelmingly dominated by processing delays, which are best spread uniformly across servers.

Using the notion of preferred quorums, we propose the following algorithm called Sticky. Whereas Balanced and ClosestDly are per-client access strategies, Sticky is, in a sense, a per-object access strategy. In Sticky, each object is accessed at its preferred quorum by default, and preferred quorums are placed so as to balance the load on each server given the anticipated workload. More specifically, Sticky is defined by a mapping $\mathsf{pref}()$ where $\mathsf{pref}(o)$ is the preferred quorum for object o. A workload induces a distribution π on objects (i.e., $\sum_o \pi(o) = 1$), in the sense that o is accessed in $\pi(o)$ fraction of the accesses. Therefore, the load on a server s, given a mapping $\mathsf{pref}()$ and workload π is,

$$\ell(s) = \sum_{o : s \in \mathsf{pref}(o)} \pi(o).$$

For any such anticipated distribution π on object accesses, Sticky is defined by a mapping $\mathsf{pref}()$ that minimizes

$$\max_{s,s'} |\ell(s) - \ell(s')|,$$

i.e., the imbalance between the most loaded and least loaded servers.

To evaluate this, we run an available version of Q/U that is updated to facilitate wide-area experiments as a result of our work (Q/U v1.3). This version of Q/U allows us to create a much larger number of clients using a small number of client machines, which is important for reasons described next.

10.2 Evaluation Challenges

When we evaluate the system, we want to be certain that we can accurately identify the cause of any performance limitations. In our evaluation we emulate a wide area network. If our evaluation setup (including the number and speed of clients; the performance of the network emulator; or the performance of the network hardware on the rack) were to represent a bottleneck, then we might be unable to draw meaningful conclusions concerning how the system would behave when actually deployed on a wide area network. Therefore, in this section, we address potential bottlenecks that might otherwise go unnoticed.

10.2.1 Increased Clients with Latency

We measure the throughput of a service as the number of requests that it processes per unit time. In general, the maximum throughput of the service is limited by either the rate at which it can process requests, or the rate at which it receives them.

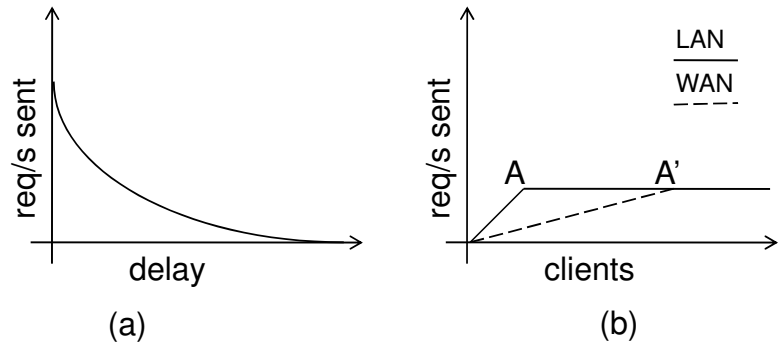


Figure 10.1: (a) Requests per second sent by a single client as a function of delay (e.g., network or processing). (b) Aggregate requests per second sent by all clients of a TCP service.

A typical strategy that is used to benchmark the peak throughput of a service (i.e., the maximum request rate that the service can handle) on a LAN is to run a number of client processes concurrently. Each client process issues a request to the service and waits for a response before immediately sending another request. Therefore, as shown in Figure 10.1 (a), the rate at which a given client issues requests is inversely proportional to the time it waits before receiving a response.

With sufficiently many client processes, enough requests can be generated by the clients to drive the service to its capacity. In the graph of the requests per second sent by clients in Figure 10.1 (b), the peak capacity of the service corresponds to point A. If the service were to receive a higher request rate, it would no longer be able to process requests as fast as they arrive.

The service is unable to process requests any faster at peak capacity. Instead, as described by Banga and Druschel [13], when a service uses TCP/IP for communication, the network stack automatically queues any requests that exceed the capacity of the service. As a result, each request takes longer to be processed, and so each client issues requests at a slower rate. In aggregate, the total request rate remains the same as the service processing rate. It does not increase unless the number of clients far exceeds the capacity of the service [13]. However, the per-request response time (i.e., latency) is increased because each client process issues requests at a slower rate.

On a wide area network, the additional network latency causes the curve in Figure 10.1 (b) to shift as shown, such that the peak request rate occurs at point A'. This is because each client process still waits until it has received a response for its current request before sending a new request. Each request takes (much) longer due to the latency, and so each client process issues requests at a (much) lower rate. This means that we require more client processes in order to provide the maximum request rate that the service can handle. In other words, as latencies increase, so too does the number of client processes necessary for benchmarking the capacity of the service.

For the hypothetical service in Figure 10.1 (b), additional network latency does not mean that the maximum throughput of the service is less; it just means that the simple benchmarking scheme requires more client processes. On the other hand, if the increase in network latency slows the processing of each request, the maximum throughput of the service may also be lower.

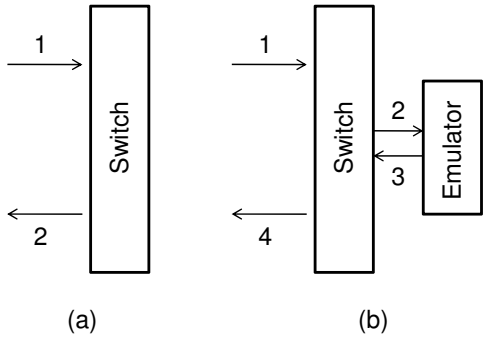


Figure 10.2: Doubling of the traffic due to the emulator.

10.2.2 Limited Emulated Bandwidth

An emulated wide-area network introduces an additional problem—the emulator and switch can themselves represent bottlenecks, particularly when emulating a network that has high bandwidth.

When a packet is sent over an emulated network, the job of the network emulator is to intercept the packet, delay (or drop) it if needed, and then forward the packet to the destination host. In a simple switched network without an emulator, each packet is forwarded directly by the switch as seen in Figure 10.2 (a). However, in an emulated network as depicted in Figure 10.2 (b), the packet is routed to the emulator and back to the switch before it is sent to its destination. As such, the switch processes roughly twice the number of packets that it would otherwise process. Even if the emulator is configured to drop traffic, then that traffic will most likely be sent again by the source, resulting in even more packets.

The network switch used for experiments has a maximum throughput capacity. Assume, for example, that the capacity of a switch is 1300 Mb/s. If an experiment that is run on the LAN consumes 650 Mb/s, the same experiment run through an emulator of sufficient capacity will consume the entire capacity of the switch. Likewise, the capacity of the emulator is limited to about half of what might be expected. This is because the emulator processes two packets for each one sent by an end host. As such, half of its bandwidth is effectively unavailable.

For example, assume that we have three identical machines. One machine acts as an emulator, and the other two act as end hosts. Each machine can process 700 Mb/s using its single gigabit network card. Without the emulator, the two end hosts could transfer 700 Mb/s between each other. However, the hosts would be able to send only roughly 350 Mb/s between each other over the emulated network. The emulator would be a bottleneck.

Some emulators like Modelnet can be configured to use multiple servers for emulation. However, one still must be careful to avoid exceeding the capacity of the switch. This effectively limits the bandwidth that can be emulated.

10.3 Evaluation

Since Q/U is currently of practical interest for small numbers of servers, we emulate a hypothetical six-server topology based on round-trip times in the USA. We configure a new Modelnet [69] topology with data from

Table 10.1, which is based closely on actual network delay measurements derived from AT&T [28]. We deploy this topology on a rack of machines with the following characteristics. The rack consists of 76 Intel Pentium 4 2.80 GHz computers, each with 1 GB of memory and an Intel PRO/1000 NIC. We use one machine as the WAN emulator and the remaining 75 as normal hosts (servers or clients).

As seen in Table 10.1, our topology consists of six network sites (San Francisco, Denver, Chicago, Cleveland, New York City, and Cambridge) located across the USA with round-trip latencies between sites as given in the table. Each site hosts one server. All clients are connected to Cambridge with zero latency and no specified bandwidth restrictions. There is one server per site, and each server is connected to its site by two unidirectional links each with 15Mbps bandwidth and zero latency. This bandwidth restriction decreases the maximum throughput of Q/U from approximately 25,000 req/s on our hardware to about 5000 req/s, as seen below, bringing the amount of data sent per second into a range that Modelnet (which must forward all of the network traffic for both clients and servers) can support on our hardware. Given that this bandwidth allocation is for a single service (i.e., not for the entire site), we feel it is reasonable. Furthermore, based on experimentation with other bandwidth levels, we expect that our results generalize. Because each client is situated in Cambridge, the closest quorum is the one that omits the server in San Francisco. This quorum requires at least 46 milliseconds round-trip latency to access. Any quorum that contains the server in San Francisco requires at least 76 milliseconds latency.

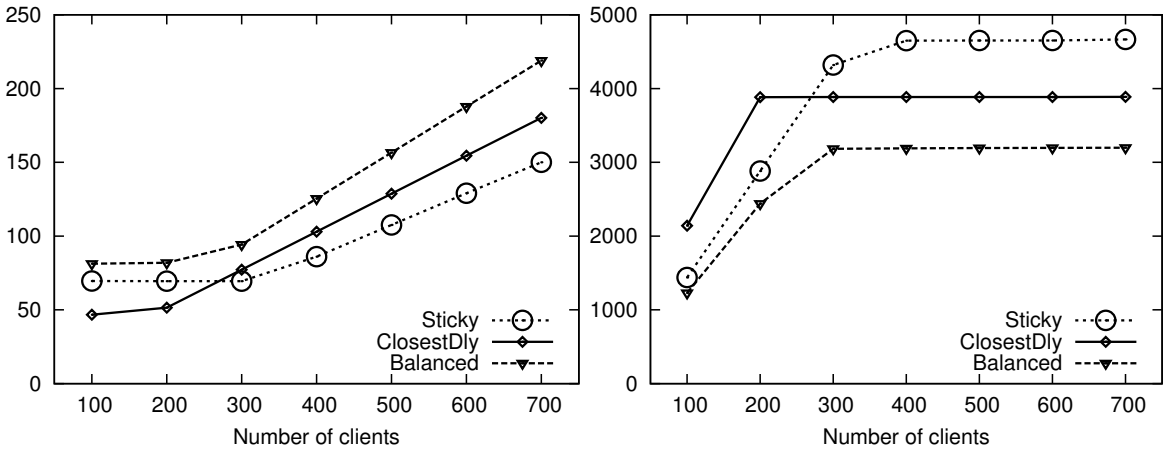
In our experiments, each client is a thread with exclusive access to its own object maintained by the service. Each client submits a request (of trivial size) to modify the object and waits for a response. Upon receiving a response that the object has been updated, the client immediately issues another modify request. As such, the workload is uniform. (Later, we address issues that may arise from other workloads.) As described in Section 10.2, because clients wait to issue requests, requests are issued at the rate at which responses are received. It takes at least the network round-trip latency for each client to receive a response. Therefore, when the network latency is longer, more clients are required to drive the service to its peak capacity. For the purposes of our evaluation, low load can be thought of as any number of clients insufficient to drive the service to its peak capacity, while high load as any number of clients achieving peak capacity. If the client has selected a different quorum for the subsequent request, the state of the object must be transferred to the server that is only in the new quorum before that server can update the object.

Because of state transfer, as Figures 10.3(a) and 10.3(b) show, ClosestDly indeed outperforms Balanced given this network configuration. However, in the presence of state transfer, Sticky behaves much as Balanced does with no state transfer. In particular, Sticky outperforms ClosestDly in periods of high load, while ClosestDly performs better than Sticky when there is low load.

To understand the impact of network latency, we repeat the above analysis, but with zero latency between sites (instead of the values from Table 10.1). The results are given in Figures 10.4(a) and 10.4(b). As

	den	**chi**	**cle**	**nyc**	**cam**
sfo	30	50	56	70	76
	den	20	26	40	46
		chi	6	20	26
			cle	14	20
				nyc	6
					cam

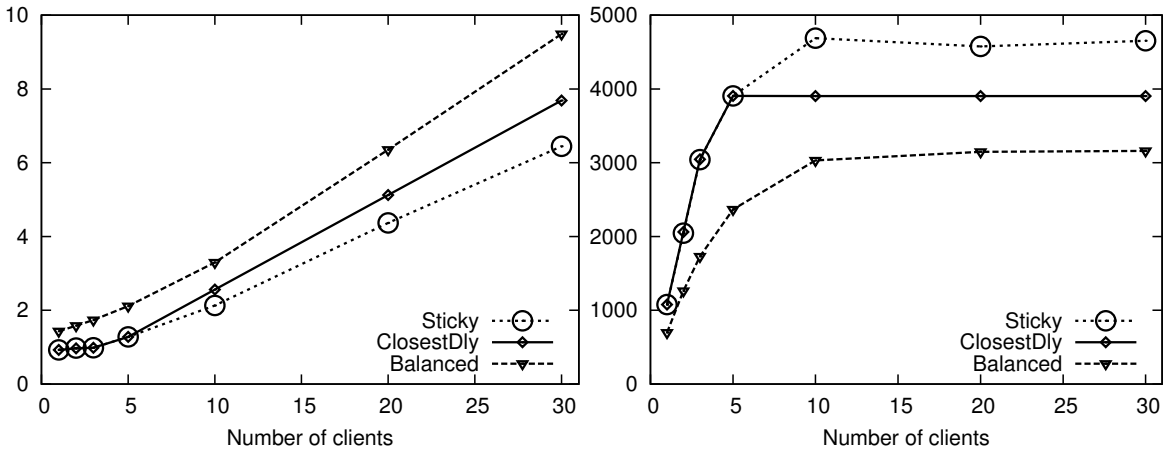
Table 10.1: Round trip network latency data (ms).



(a) Response time (ms) (b) Throughput (req/s)

Figure 10.3: Q/U with state transfer.

expected, because all quorums are equally close, Sticky and ClosestDly perform equally well here when there is low load. However, Sticky again outperforms ClosestDly when there is high load. In all cases, as before, Balanced performs worse than either Sticky or ClosestDly because of its need for state transfer.



(a) Response time (ms) (b) Throughput (req/s)

Figure 10.4: Q/U with state transfer, no delay.

The use of Sticky as a substitute for Balanced requires a way to move preferred quorums. A sketch of a conceptually simple modification to Q/U in order to do so is as follows. First, note that we can use Q/U itself to maintain the location of the preferred quorum for each object. Therefore, to update the preferred quorum, a client performs an update operation specifying the new preferred quorum. This operation for updating an object's preferred quorum is performed like any other (linearizable) operation for the service, so that it is unambiguous as to what the preferred quorum is at any logical point in time. When the object is updated,

the operation can be specified as *contingent* on the selected quorum being the preferred quorum (to maintain efficiency by avoiding state transfer), or as *without contingencies* (which might cause state transfer). In the former case, servers reject the operation if the selected quorum is not the preferred quorum for the accessed object. They do this on the basis of state information, included with the operation, that is relayed by clients from servers to servers (called object history sets [1] in Q/U terminology) for the preferred quorum. In such a case, the client must resubmit the operation.

To learn the preferred quorum, a client uses a query operation specifying the object. Although the client can use a query to lookup the preferred quorum for an object, optimizations such as the following minimize such lookups (so as to minimize the number of rounds of interaction on the critical path of an operation). A client caches the preferred quorum following each lookup, and performs its subsequent operations using contingent requests. Upon servers rejecting this operation due to the preferred quorum changing, the client can be informed in the rejection of the new preferred quorum, or it can lookup the preferred quorum again. The client issues requests without contingencies when the preferred quorum is unresponsive.

Sticky maximizes throughput and minimizes average response time when there is high load (as does Balanced when there is no state transfer). Despite this, if multiple clients access the same object, the placement of a given preferred quorum can be better in terms of individual response times for certain clients than for others (e.g., when some clients are close to the preferred quorum while others are far from it). This may give incentive to clients to move preferred quorums close to themselves. However, the benefits of Sticky compared with Balanced can be negated without well-defined policies for updating preferred quorums. For example, a client that changes the preferred quorum to be closer to itself may cause state transfer, both now and again later if another client moves the preferred quorum back. It remains an open question to develop policies for changing preferred quorums for any particular workload.

10.4 Summary

The take-away message from the analysis in this chapter is that state transfer can have an impact on the performance of any algorithm that switches quorums. In such situations, algorithms such as Sticky that balance load but avoid state transfer should be considered as a substitute for Balanced.

Chapter 11

Related Work

11.1 Probabilistic Quorum Systems.

A Probabilistic Quorum System (PQS), as presented by Malkhi et al. [47], can provide better availability and fault tolerance than strict quorum systems can provide. Malkhi et al. introduce constructions for dissemination and masking quorums, and prove properties of load and availability for these constructions. They neither address opaque quorum systems, nor the effects of concurrent or Byzantine writers; we address each of these. In addition, we borrow analysis techniques from [47], but our analysis is more general in the sense that clients are not all assumed to communicate only with quorums of servers. We also use a McDiarmid inequality [50] for bounding the error probability; this provides a simpler bounding technique for our purposes than do the Chernoff bounds used there. The technique that we present in Chapter 8 for restricting access to limited numbers of servers should be applicable to the constructions of Malkhi et al. equally well.

Probabilistic quorum systems were explored in the context of dynamic systems with non-uniform access strategies by Abraham and Malkhi [2]. Recently, probabilistic quorum systems have been used in the context of security for wireless sensor networks [25] as well as storage for mobile ad hoc networks [41]. Lee and Welch make use of probabilistic quorum systems in randomized algorithms for distributed read-write registers [38] and shared queue data structures [39].

11.2 Opaque Quorum Systems.

Opaque Byzantine quorum systems were introduced by Malkhi and Reiter [43] in two variants: one in which the number of non-faulty servers in a quorum is at least half of the quorum; and the other in which the number of non-faulty servers represents a strict majority of the quorum. The first construction makes it unnecessary for the client to know the sets of servers of which the system can tolerate failure (hence the term 'opaque'), while the second construction additionally makes it possible to create a protocol that does not use timestamps. The paper also proves that $5b$ is the lower bound on the number of servers for the first version; simply changing the inequality to a strict inequality proves $5b+1$ is the lower bound for the second. In this paper, when we refer to strict (non-probabilistic) opaque quorum systems, we are concerned with the second variant.

The constraints on strict opaque quorums have also been described in the context of consensus and state-machine-replication protocols, e.g., the Q/U [1] and FaB Paxos [48] protocols, though not explicitly

as opaque quorums. Abd-El-Malek et al. [1] provide generic (not just threshold) opaque quorum system constraints that they prove sufficient for providing state-machine replication semantics where both writes and reads complete in a single (pipelined) phase when there is no write–write contention. Martin and Alvisi [48] use an opaque quorum system of acceptors in FaB Paxos, a two-phase consensus protocol (with a designated proposer) and three-phase state-machine-replication protocol requiring at least $5b + 1$ servers.

11.3 Signed Quorum Systems

Signed Quorum Systems [72] are another attempt to weaken the requirements of strict quorum systems. A quorum in a signed quorum system can include both servers that respond and servers that are polled but do not respond (and are, therefore, believed by the client to have crashed); if a server responds in one quorum but is marked as crashed in a different quorum, the quorums are said to *mismatch* for that server. A signed quorum system is constructed such that if any two quorums do not overlap in a server that responds to both quorum accesses, the quorums must have at least 2α mismatches (this is known as the *dual-overlap* property). Then, given the assumptions that it is rare for any two clients to see a mismatch for a given server, i.e., that a mismatch occurs with probability at most ϵ, and that the probability of a mismatch for a given server is independent of that for any other server, the probability of two quorums not overlapping (and hence mismatching in at least 2α servers) is at most $1 - \epsilon^{2\alpha}$. This assumption does not lend itself directly to the Byzantine fault model in which a faulty server may choose to respond to one client but not to another. Thus, while signed quorums are related to probabilistic quorums, they are not presently affected by write markers because they have not been studied in the context of Byzantine faults. Here, in our analysis of probabilistic quorums, we find that tolerance of Byzantine faults substantially alters both the analysis techniques needed and the outcomes that result.

11.4 K-Quorums

k-quorums [6] also weaken the requirements of strict quorum systems in an effort to provide greater availability, but focus on offering a property called *bounded staleness* that ensures (with certainty, as opposed to with high probability) that a read will receive one of the last k writes, even if messages may be delivered according to the choices of an unconstrained adversary. This is achieved by requiring that the union of the last k writes intersects any read quorum. k-quorums have been extended to support Byzantine-faulty servers, and multi-writer protocols [7]. However, as we are not concerned with the bounded staleness property that is central to k-quorums, our results are orthogonal and different from those. Moreover, our results include treatment of Byzantine-faulty clients.

11.5 Tolerating Byzantine Clients

No prior work on any of the three types of non-strict quorum systems listed above (probabilistic quorums, signed quorums, or k-quorums) considers Byzantine clients. There has been work on strict quorum systems that can tolerate Byzantine clients (e.g., [40, 19]) but this is fundamentally unconcerned with the way in which quorums are chosen because such choices cannot impact the correctness of strict quorum systems.

11.6 Write Markers

Another implementation of write markers was introduced by Alvisi et al. [8] for purposes different than ours. By using write markers to prevent some faulty servers from colluding, we achieve the goals of: (i) improving the load; and (ii) increasing the maximum fraction of faults that the system can tolerate. In contrast to this, Alvisi et al. use write markers in order to increase accuracy in estimating the number of faults present in Byzantine quorum systems, and for identifying faulty servers that consistently return incorrect results. Because the implementation of Alvisi et al. does not prevent faulty servers from lying about the write quorums of which they are members, it cannot be used directly for our purposes. In addition, our implementation is designed to tolerate Byzantine clients, unlike theirs.

11.7 Other Scalability Approaches

Q/U [1] and H/Q [23] are so-called *fault-scalable* protocols that use client-server communication (in the form of quorum accesses using threshold quorum systems) instead of server-server communication such as in BFT [22]. Thus, Q/U and HQ provide $O(n)$ instead of $O(n^2)$ communication complexity in the number of servers. Moreover, because Q/U and HQ are explicitly quorum based, they have the efficiency and load benefits gained by not accessing all n servers.

Another approach to scalability is to allow the system to be partitioned into multiple services or objects, each of which can tolerate a small number of faults [59, 45, 65, 27, 51, 9]. In general, this changes the fault tolerance of the system—too many faults in any partition can compromise the entire system. Representative examples of this approach are as follows. OceanStore [62] is designed to be an Internet-scale persistent data store, while FARSITE [3] is a secure, scalable file system. To achieve scalability without a fault-scalable protocol, the use of Byzantine fault tolerance is limited to parts of each system (the primary replica of each object in OceanStore, and directory groups in FARSITE). Similarly, the BAR protocol [5] uses a Byzantine fault tolerant protocol to create primitives implemented by a subset of the nodes that provide incentives for rational nodes to follow the protocol. Thema [51] can be used to enable Web services to tolerate Byzantine faults on an individual basis, yet still communicate with other Web services. Finally, two multi-tiered Byzantine-fault-tolerant DNS systems have been presented [70, 4], both of which use the BFT protocol to replicate each node in the DNS hierarchy individually.

Non-threshold dissemination and masking quorum systems have been introduced [43, 46]. These quorum systems, such as grid quorum systems and path quorum systems, generally have better load, but worse availability and fault tolerance, than threshold quorum systems. This is because non-threshold quorum systems specify groups of servers for quorums, instead of allowing any group of a given size to be a quorum as in threshold quorum systems. In a sense, probabilistic quorum systems lie between these two approaches. In a probabilistic quorum system, any group of a given size represents a quorum, but only if all of the servers are contained in a randomly chosen access set. By increasing the size of the access sets, we therefore improve availability at the expense of fault tolerance as shown in Chapter 6. Alternatively, we can allow clients to choose from multiple access sets for better availability, but, then, measures, such as those discussed in Chapter 8, must be taken to ensure that we do not compromise consistency as a result.

11.8 Distributed Hash Tables

Distributed hash tables (DHTs) such as Chord [68], Pastry [64], and Tapestry [73], are structured overlay networks that provide a way for a node to locate content in a peer-to-peer network. Like a hash table, data is identified by keys, and there is a mapping from keys to likely locations for the data. The focus is on efficiently being able to route a message to a node or set of nodes responsible for the data corresponding to the key. The responsibility for (and location of) content is dynamic. The content itself can be replicated amongst many nodes for fault tolerance. DHTs are designed for scalability in the sense that no node needs global knowledge of the system. Instead of updating the state of every node for each change (potentially expensive in a large system), nodes keep a small amount of state and make local routing decisions based on that state. Though nodes may join or leave the network, DHTs ensure that enough nodes remain responsible for content and the routes to that content. The systems are decentralized, and all nodes have identical responsibilities.

As a representative example, consider Pastry [64]. The data is managed by the k nodes with identifiers closest to the key. Nodes maintain a small amount of local routing state. Routes take an expected $O(\log(n))$ hops to reach the destination, and delivery is guaranteed so long as k neighbors are not simultaneously unreachable. Node identifiers are assigned randomly, so neighbors are less likely to be impacted by correlated failures. Pastry supports locality, so a message is likely to be delivered along an efficient path (in terms of some distance metric).

Probabilistic quorum systems and DHTs can both have low load but for different reasons. In a probabilistic quorum system, the membership of the system is known by the clients and so "lookup" for any data is a constant time operation—a client simply contacts the servers in a quorum directly. The quorums are of size at least $O(\sqrt{n})$. In a DHT, the membership of the system is not known by each node. Instead, lookup is expected to involve $O(\log(n))$ nodes. In probabilistic quorum systems, quorums of nodes store each value, while in a DHT, k randomly chosen nodes store each value. In general, DHTs are not designed to tolerate Byzantine faults, but some work has focused on how to do so as discussed next.

Castro et al. [21] focus on how to make a DHT tolerant to Byzantine faults. They identify *secure routing* as the basic primitive. Secure routing guarantees that, with high probability, a message sent by a non-faulty node reaches the non-faulty nodes in the set of the k nodes responsible for the key. The primary challenges are: ensuring that the message is eventually delivered even though faulty nodes may attempt to thwart this; and ensuring that the message is delivered to the legitimate targets of the message (i.e., not to nodes that impersonate the targets). They propose that secure routing can be achieved by: (i) securely assigning identifiers to nodes, e.g., by using a centralized certification authority; (ii) securely maintaining the routing tables, e.g., by having the option to use less efficient but constrained routing tables that limit the percentage of faulty nodes in the routing tables with high probability; and (iii) securely forwarding messages, e.g., by detecting faults and using diverse routes to ensure that the routing protocol is followed with high probability. In their analysis, they present a variation on the Byzantine fault model that allows for restricting the size of groups of faulty nodes that can collude to $c < b$.

Baden et al. [12] seek to provide a Byzantine-fault-tolerant DHT that addresses the Sybil attack. They allow an adversary to control an arbitrary fraction of the nodes, but only nodes that have one of f attributes from a larger set. For example, assume the attacker can successfully compromise at most two operating systems (i.e., $f = 2$). The hosts are separated into at least $f + 1$ node-independent DHTs by attributes. Continuing the example, each DHT therefore contains only nodes with a specified operating system. The

$f + 1$ DHTs (including at least one that is correct because all of its nodes are non-faulty) are accessed in parallel and so the system is deemed secure.

Chapter 12

Conclusions

12.1 Summary

Byzantine fault tolerance can provide survivability for networked services. Byzantine-fault-tolerant replication protocols for the services build on Byzantine quorum systems in order to provide consistency and availability. Characteristics of the quorum systems such as fault tolerance, load, and availability impact the corresponding characteristics of the service protocols because of the relationship between the service protocols and quorum systems that is shown in Chapter 2.

In this dissertation, we have focused primarily on *probabilistic* quorum systems as a way to achieve better fault-tolerance and load properties. While probabilistic quorum systems admit a bounded probability of inconsistency, they improve upon the fault-tolerance properties of dissemination, masking, and opaque quorum systems. Moreover, if implemented with the *write markers* that we have introduced, they improve upon the load properties.

This dissertation also furthers the understanding of probabilistic quorum systems in additional ways including: introduction of a probabilistic opaque quorum system without write markers that improves upon the fault tolerance of opaque quorum systems; introduction of a framework for proving properties of probabilistic quorum systems that enables our analysis of write markers and opaque systems, but also proves that masking systems can tolerate more faults than shown previously; a way to improve the availability of probabilistic quorum systems through the use of access sets in addition to quorums; a model of how Byzantine-faulty clients can seek to undermine the properties of probabilistic quorum systems; a protocol that allows probabilistic quorum systems to tolerate Byzantine-faulty clients; and an analysis of the costs of switching quorums when a protocol maintains state that must be transferred between replicas.

12.2 Future Work

Our work in probabilistic quorum systems should provide the ability to use Byzantine quorum systems at a larger scale than previously feasible. In particular, we can: (i) provide lower load and/or (ii) tolerate more Byzantine faults than can strict quorum systems. The lower load gives a way to use the servers more efficiently, as well as allowing for smaller quorums that can be accessed more efficiently by clients. Tolerating more faults suggests that we can: (a) use a smaller system of servers for the same fault tolerance;

and/or (b) make a more reliable system for the same number of servers. However, the benefits of our approach come at the cost of a (bounded) probability of error.

Three factors should be considered when assessing the suitability of probabilistic quorum systems for a given application. First, probabilistic quorum systems are best able to achieve low error probabilities when the replication factor is high. Second, the access strategy requires frequent quorum changes, which may result in state-transfer costs for certain types of applications as described in Chapter 10. Third, there is an additional overhead in the form of an access-restriction mechanism such as that described in Chapter 8 if faulty clients must be tolerated. We first consider applications that may benefit from probabilistic quorum systems as they currently exist, and then suggest research directions for addressing these three concerns.

Potential applications. Probabilistic quorum systems, especially those with write markers, may be particularly well-suited for Byzantine-fault-tolerant read-overwrite protocols for storage like the PASIS-RW protocol [29] discussed in Chapter 2. This is because such protocols: (i) may derive fault-tolerance benefits from having large numbers of replicas; (ii) do not necessarily involve state transfer between replicas because the client explicitly overwrites the old data at a new quorum; and (iii) may have no need to tolerate faulty clients in cases such as when each client has exclusive access to its own files and therefore cannot impact other clients with any inconsistency in its files. Yet, when highly replicated, they should be able to take advantage of the increased efficiency in the form of lower load, as well as the increased fault tolerance, provided by probabilistic quorum systems with write markers.

If the overhead from state transfer is deemed acceptable for a particular application, then quorum-based state-machine-replication protocols like Q/U [1] and HQ [23] might also benefit from the increased fault tolerance and reduced load afforded by probabilistic quorum systems, particularly those with write markers. For example, many distributed systems rely on membership services for determining the set of participants in the system. Such membership services can represent a single point of failure. Probabilistic quorum systems could be used in this context to provide a highly-replicated, global, shared membership service implemented with state machine replication.

Finally, probabilistic quorums with write markers might be combined with strict quorums with useful results. For example, consider a frequently-read, but rarely-updated Byzantine-fault-tolerant data store with a single writer. The writer could use a strict quorum system to update the servers with no chance of inconsistency, while clients could use probabilistic quorums for reads that are more efficient (albeit with a probability of inconsistency) due to the smaller quorum sizes afforded by the lower load.

Addressing the Error Probability. Error probability calculations such as those given here and in [47] are pessimistic estimates because they assume that all faulty clients and replicas try to cause errors. Moreover, they assume that multiple operations (such as a read and a write) occur in very rapid succession. However, in real workloads, faults may not manifest themselves at all times. Moreover, there may be time between operations during which the error probability can be reduced using techniques such as the background propagation discussed in Chapter 8. Thus, one avenue of research is to design and evaluate ways to make the actual error probability for real workloads lower than the calculated error probabilities.

Addressing Costs of State Transfer. State transfer is viewed as a rare event both in systems like BFT [22] and Zyzzyva [35] that attempt to involve all replicas in every operation in the common case, and in systems like Q/U that use preferred quorums. Probabilistic quorum systems, on the other hand, explicitly seek to

communicate each operation only to a small set of servers (for the sake of efficiency), but to a set that changes membership frequently because of the access strategy (for the sake of consistency and low load). Thus, probabilistic quorum systems could benefit from a way to minimize the perceived overheads of state transfer. When client updates are infrequent, one approach could be the background dissemination of new state discussed in Chapter 8 so that the cost of state transfer is incurred without slowing down the next operation. For more frequently updated systems, a way to offload the burden of state transfer from the servers to the clients could promote increased scalability and throughput.

Enforcing the Access Strategy. Because all clients must follow the access strategy but faulty clients cannot be trusted to do so on their own, the protocol of Chapter 8 involves the non-faulty replicas. The protocol is designed to amortize any resultant additional communication between a client and the replicas over multiple access sets. However, it must balance this goal with preventing clients from choosing from multiple access sets for a given operation. Therefore, it involves techniques such as threshold signatures, non-interactive client puzzles, and background propagation. Many alternative designs might become available given different constraints, such as if the servers can be involved more frequently.

Bibliography

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable Byzantine fault-tolerant services. In *Symposium on Operating Systems Principles*, October 2005.

[2] I. Abraham and D. Malkhi. Probabilistic quorums for dynamic systems. *Distributed Computing*, 18(2):113 – 124, 2005.

[3] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Operating Systems Design and Implementation*, 2002.

[4] S. Ahmed. A scalable Byzantine fault tolerant secure domain name system. Master's thesis, MIT, Jan. 2001. Also as Technical Report MIT-LCS-TR-849.

[5] A. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *Symposium on Operating Systems Principles*, pages 45–58, 2005.

[6] A. S. Aiyer, L. Alvisi, and R. A. Bazzi. On the availability of non-strict quorum systems. In *DISC 2005*, pages 48–62, 2005.

[7] A. S. Aiyer, L. Alvisi, and R. A. Bazzi. Byzantine and multi-writer k-quorums. In *DISC 2006*, pages 443–458, 2006.

[8] L. Alvisi, D. Malkhi, E. Pierce, and M. K. Reiter. Fault detection for Byzantine quorum systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):996–1007, Sept. 2001.

[9] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Scaling Byzantine fault-tolerant replication to wide area networks. In *International Conference on Dependable Systems and Networks*, 2006.

[10] J. H. An, Y. Dodis, and T. Rabin. On the security of joint signature and encryption. In *EUROCRYPT 2002*, pages 83–107, London, UK, 2002.

[11] A. Back. Hashcash - a denial of service counter-measure. `http://cypherspace.org/hashcash/hashcash.pdf`, August 2002.

[12] R. Baden, A. Bender, D. Levin, R. Sherwood, N. Spring, and B. Bhattacharjee. A secure DHT via the pigeonhole principle. Technical Report TR-CS-4884, University of Maryland, Sept. 2007.

[13] G. Banga and P. Druschel. Measuring the capacity of a Web server under realistic loads. *World Wide Web*, 2(1-2):69–83, 1999.

[14] R. A. Bazzi. Access cost for asynchronous Byzantine quorum systems. *Distributed Computing*, 14(1):41–48, 2001.

[15] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Conference on Computer and Communications Security*, pages 62–73, 1993.

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

[17] C. Cachin and J. A. Poritz. Secure intrusion-tolerant replication on the Internet. In *International Conference on Dependable Systems and Networks*, pages 167–176, 2002.

[18] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *Symposium on Reliable Distributed Systems*, 2005.

[19] C. Cachin and S. Tessaro. Optimal resilience for erasure-coded Byzantine distributed storage. In *International Conference on Dependable Systems and Networks*, 2006.

[20] M. Castro. *Practical Byzantine Fault Tolerance*. PhD thesis, MIT, 2001.

[21] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Symposium on Operating Systems Design and Implementation*, pages 299–314, 2002.

[22] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002.

[23] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Symposium on Operating Systems Design and Implementation*, Nov. 2006.

[24] A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Principles of Distributed Computing*, pages 1–12, August 1987.

[25] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, and A. Khalili. A pairwise key predistribution scheme for wireless sensor networks. *ACM Transactions on Information and System Security*, 8(2):228–258, 2005.

[26] R. J. Ellison, D. A. Fisher, R. C. Linger, H. F. Lipson, T. Longstaff, and N. R. Mead. Survivable network systems: An emerging discipline. Technical Report CMU/SEI-97-TR-013, CMU SEI, 1997.

[27] C. P. Fry and M. K. Reiter. Nested objects in a Byzantine quorum-replicated system. In *IEEE International Symposium on Reliable Distributed Systems*, pages 77–89, 2004.

[28] Global IP network home. `http://ipnetwork.bgtmo.ip.att.net`, December 2008.

[29] G. R. Goodson, J. J. Wylie, G. R. Ganger, and M. K. Reiter. Efficient Byzantine-tolerant erasure-coded storage. In *International Conference on Dependable Systems and Networks*, June 2004.

[30] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead Byzantine fault-tolerant storage. In *Symposium on Operating Systems Principles*, Oct. 2007.

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

[32] M. Jakobsson and A. Juels. Proofs of work and bread pudding protocols. In *Communications and Multimedia Security*, pages 258–272, 1999.

[33] A. Juels and J. Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Network and Distributed Systems Security Symposium*, pages 151–165, 1999.

[34] L. Kong, D. Manohar, A. Subbiah, M. Sun, M. Ahamad, and D. Blough. Agile store: Experience with quorum-based data replication techniques for adaptive Byzantine fault tolerance. In *IEEE Symposium on Reliable Distributed Systems*, pages 143–154, 2005.

[35] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine fault tolerance. In *Symposium on Operating Systems Principles*, pages 45–58, New York, NY, USA, 2007. ACM.

[36] R. Kotla and M. Dahlin. High throughput Byzantine fault tolerance. In *International Conference on Dependable Systems and Networks*, page 575, June–July 2004.

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

[38] H. Lee and J. L. Welch. Applications of probabilistic quorums to iterative algorithms. In *International Conference on Distributed Computing Systems*, pages 21–30, Apr. 2001.

[39] H. Lee and J. L. Welch. Randomized shared queues applied to distributed optimization algorithms. In *International Symposium on Algorithms and Computation*, December 2001.

[40] B. Liskov and R. Rodrigues. Tolerating Byzantine faulty clients in a quorum system. In *International Conference on Distributed Computing Systems*, 2006.

[41] J. Luo, J.-P. Hubaux, and P. T. Eugster. Pan: Providing reliable storage in mobile ad hoc networks with probabilistic quorum systems. In *International symposium on mobile ad hoc networking and computing*, pages 1–12, 2003.

[42] D. Malkhi, Y. Mansour, and M. K. Reiter. Diffusion without false rumors: On propagating updates in a Byzantine environment. *Theoretical Computer Science*, 299(1–3):289–306, 2003.

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

[44] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202, Apr. 2000.

[45] D. Malkhi, M. K. Reiter, D. Tulone, and E. Ziskind. Persistent objects in the Fleet system. In *DARPA Information Survivability Conference & Exposition*, volume 2, pages 126–136, 2001.

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

[47] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic quorum systems. *Information and Computation*, 170(2):184–206, 2001.

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

[49] J.-P. Martin, L. Alvisi, and M. Dahlin. Minimal Byzantine storage. In *International Symposium on Distributed Computing*, 2002.

[50] C. McDiarmid. Concentration for independent permutations. *Combinatorics, Probability and Computing*, 11(2):163–178, 2002.

[51] M. G. Merideth, A. Iyengar, T. Mikalsen, S. Tai, I. Rouvellou, and P. Narasimhan. Thema: Byzantine-fault-tolerant middleware for web-service applications. In *Symposium on Reliable Distributed Systems*, Oct 2005.

[52] M. G. Merideth, F. Oprea, and M. K. Reiter. When and how to change quorums on wide area networks, 2008. (*under submission*).

[53] M. G. Merideth and M. K. Reiter. Probabilistic opaque quorum systems. In *International Symposium on Distributed Computing*, Sept. 2007.

[54] M. G. Merideth and M. K. Reiter. Write markers for probabilistic quorum systems. In *International Conference on Principles of Distributed Systems*, Dec. 2008.

[55] M. G. Merideth and M. K. Reiter. Selected results from the latest decade of quorum systems research, 2009. (*to appear as book chapter*).

[56] M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2005.

[57] M. Molloy and B. Reed. *Graph Colouring and the Probabilistic Method*. Springer, 2002.

[58] M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.

[59] P. Narasimhan, K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Providing support for survivable CORBA applications with the Immune system. In *International Conference on Distributed Computing Systems*, pages 507–516, 1999.

[60] F. Oprea and M. K. Reiter. Minimizing response time for quorum-system protocols over wide-area networks. In *International Conference on Dependable Systems and Networks*, June 2007.

[61] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Conference on Computer and Communication Security*, pages 68–80, November 1994.

[62] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao, and J. Kubiatowicz. Pond: The OceanStore prototype. In *Conference on File and Storage Technologies*, 2003.

[63] R. Rodrigues, M. Castro, and B. Liskov. BASE: Using abstraction to improve fault tolerance. In *Symposium on Operating Systems Principles*, 2001.

[64] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware*, Nov. 2001.

[65] D. Sames, B. Matt, B. Niebuhr, G. Tally, B. Whitmore, and D. Bakken. Developing a heterogeneous intrusion tolerant CORBA system. In *International Conference on Dependable Systems and Networks*, pages 239–248, 2002.

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

[67] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15(2):75–96, 2002.

[68] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM*, pages 149–160, Aug. 2001.

[69] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. *SIGOPS Oper. Syst. Rev.*, 36(SI):271–284, 2002.

[70] Z. Yang. Using a Byzantine-fault-tolerant algorithm to provide a secure DNS. Master's thesis, MIT, Jan. 1999.

[71] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *Symposium on Operating Systems Principles*, pages 253–267, October 2003.

[72] H. Yu. Signed quorum systems. *Distributed Computing*, 18(4):307–323, 2006.

[73] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California, Berkeley, Apr. 2001.