Fault Detection for Byzantine Quorum Systems

Lorenzo Alvisi* Dahlia Malkhi† Evelyn Pierce‡ Michael K. Reiter§

Abstract

In this paper we explore techniques to detect Byzantine server failures in asynchronous replicated data services. Our goal is to detect arbitrary failures of data servers in a system where each client accesses the replicated data at only a subset (quorum) of servers in each operation. In such a system, some correct servers can be out of date after a write and can therefore return values other than the most up-to-date value in response to a client's read request, thus complicating the task of determining the number of faulty servers in the system at any point in time. We initiate the study of detecting server failures in this context, and propose two statistical approaches for estimating the risk posed by faulty servers based on responses to read requests.

Index terms: Byzantine fault tolerance, replicated data, quorum systems, fault detection

*Department of Computer Science, University of Texas, Austin, Texas; lorenzo@cs.utexas.edu.
†Institute of Computer Science, The Hebrew University of Jerusalem, Israel; dalia@cs.huji.ac.il.
‡Department of Computer Science, University of Texas, Austin, Texas; tumlin@cs.utexas.edu.
§Bell Labs, Lucent Technologies, Murray Hill, New Jersey; reiter@research.bell-labs.com.

This document describes work supported by NSF CAREER award CCR-9734185, DARPA/SPAWAR grant N66001-98-8911, NSF CISE grant CDA-9624082, and United States Air Force contract F30602-99-C-0165. Any opinions, findings, conclusions or recommendations expressed in this document are those of the author(s) and do not necessarily reflect the views of any of these sponsoring organizations. A preliminary version of this paper appears as [AMPR99].

1 Introduction

Data replication is a well-known means of protecting against data unavailability or corruption in the face of data server failures. Several recent works have studied the use of *quorums* for replicating data efficiently across a potentially large set of data servers, and in a way that ensures that clients receive correct data even in the presence of arbitrary (Byzantine) server failures [Baz97, MR98, MRW99]. A defining property of these constructions is that each update to the data is sent to only a subset, or quorum, of the servers. Such designs pose new challenges for monitoring the number of faulty servers, since correct servers may hold different versions of the data; specifically, some correct servers may not hold up-to-date data. Thus, in a read operation, an inconsistent response from a server does not necessarily indicate the server's failure.

In this paper, we initiate the study of statistical methods for Byzantine fault detection in systems replicated among a universe U of servers using quorum methods. Specifically, we address the problem of detecting the presence (and to an extent, the identity) of servers that respond to queries with incorrect data. Our techniques are specifically designed for asynchronous systems replicated using *Byzantine quorum systems* [MR98] and *probabilistic Byzantine quorum systems* [MRWW98], although our general approach may shed light on Byzantine failure detection in other settings. A Byzantine quorum system as defined in [MR98] is designed to mask any *failure scenario* (set of faulty servers) contained within a *failure assumption* $\mathcal{B} \subset 2^U$ specified as a parameter to the quorum system construction. For example, a common failure assumption $\mathcal{B}$ is that containing all subsets of servers of at most

a specified size t; this expresses the common assumption that at most t servers fail. The goal of this work is to detect when the set of actual failures in the system is approaching an element of $\overline{\mathcal{B}}$, i.e., a failure scenario that the Byzantine quorum system is not designed to mask.

Our motivation for exploring Byzantine failure detection is drawn from a survivable and scalable data store called Fleet [MR99]. Fleet is designed to provide consistent data services in very challenging settings, where servers are dispersed over a large network and may suffer malicious penetration by attackers. Since each Fleet data object is built to mask any failure scenario in its predetermined failure assumption, it is important that the system be monitored to detect a situation in which actual failures are approaching any of the failure scenario outside this assumption. Monitoring failures can also improve the efficiency of quorum selection in Fleet. Our goal is to design detection algorithms that otherwise interfere with the system minimally.

Because this is an initial investigation into the statistical monitoring of replicated data, we simplify the problem in a few key ways. First, we perform our analysis in the context of read operations that are concurrent with no write operations, as observing partially completed writes during a read substantially complicates the task of inferring server failures. In practice (e.g, in Fleet), this is achieved using probabilistic and highly efficient locking techniques [CMR00]. Second, we assume that *clients* fail by crashing only. This restriction may seem unrealistic when servers are presumed to fail arbitrarily. However, in the context of Fleet, the creator of a data object can prohibit untrusted clients from modifying that object using access control mechanisms that remain in force even when servers fail arbitrar-

ily (provided that failure assumptions are not otherwise violated). Thus, our assumption of client crashes in practice reduces to an assumption about the clients trusted by the creator of an object to modify that object. Third, our techniques are most effective in detecting faulty servers that consistently return incorrect responses, and we restrict our attention to this case here. Faulty servers that evidence their failure only intermittently are more difficult to detect, and we leave this issue for future work.

Our technical approach is as follows: We assume that the underlying system is accessed using quorums designed to mask the failure of any failure scenario in its failure assumption $\mathcal{B}$. We set an *alarm distance* Δ with the intention of issuing a warning whenever the set of actually faulty servers F are within Δ of some element of the set $2^U \setminus \mathcal{B}$, which we call $\overline{\mathcal{B}}$; i.e., there exists a set S of Δ servers such that $F \cup S \in \overline{\mathcal{B}}$. We call such an F *dangerous*. We note that our definition does not distinguish between the case that there are are *many* such sets S, and the case where there are only a few (or even one) such sets S. When a reader obtains responses from a quorum of servers, it classifies the responding servers into those that returned answers consistent with the one it chose as the correct answer—this is called the *justifying set* for that answer—and those that did not. Our technique determines the probability that a certain observed justifying set indicates that F is dangerous, and an alarm is triggered if this probability is sufficiently high. We further enhance this approach with a generic tagging technique, called a *write marker* protocol, which tags the quorum to which an update is written, so that it can later be identified by a reader. With the write marker protocol, certain incorrect responses by faulty servers indicate without doubt that they are faulty. We again apply statistical tests to determine when such a set of detected failures

indicates an unacceptably high probability of a dangerous failure scenario. In both cases, we show that if the alarm distance is correctly selected and read operations are frequent, both methods can be expected to issue warnings in a timely fashion, i.e., within a few incorrect responses by a sufficient number of faulty servers.

An attractive feature of our approach to failure detection is that it operates within the standard read and write, by mining the responses received from servers for indication of failure. Furthermore, our approach makes it impossible for faulty servers to return systematically incorrect responses and yet avoid detection by our mechanisms: in each data read, each faulty server must either return an incorrect answer (or nothing) and risk detection, or return the correct answer.

To summarize, the contributions of this paper are twofold: we initiate the study of fault monitoring and detection in the context of quorum-replicated data; and we propose two statistical techniques for performing this detection for Byzantine quorum systems under the conditions described above. We begin by surveying related work in Section 2. In Section 3 we describe our system model and necessary background. In Sections 4–5 we present and analyze our two statistical methods using exact formulae for alarm line placement in relatively small systems. In Section 6 we present an asymptotic analysis for estimating appropriate alarm line placement in larger systems for both methods. We conclude in Section 7.

2 Related Work

Our work is most directly related to prior work on the diagnosis of faults in multiprocessor systems. Of particular relevance is the general approach introduced in [Mal80, MM81, HC81], in which the faulty or correct status of a processor is determined by comparing its responses to requests with the responses to the same requests produced by other processors. This approach has been extended for effectively diagnosing fully arbitrary faults in distributed systems (e.g., [SR87, BB93]). Like [Mal80, HC81], in which a presumed-correct system performs comparisons to discover faulty processors, this paper presumes that the clients, which solicit responses from servers and perform comparisons, behave correctly. However, our work differs from all the previous work of which we are aware in two important ways. First, whereas prior work in fault diagnosis has focused only on identifying faulty processors, we also focus on evaluating a hypothesis on the *total number* of faulty servers based on a limited probing of the system, i.e., a single quorum access. Second, our work is targeted at a setting in which even correct servers may return inconsistent values to a read query, because updates are sent to only a quorum of servers.

Somewhat more distantly related is work on intrusion detection (e.g., [Amo99]). In *host-based* intrusion detection, a trusted subsystem on each host monitors events on that host for evidence of known attack patterns or anomalous behavior. Our work differs from host-based intrusion detection in that we presume a faulty server is entirely corrupted; there is no trusted subsystem on that server to monitor and report deviant behavior. A *network-based* intrusion detection system monitors network traffic on the network segment to which

it is connected for evidence of possible attacks or anomalies. In particular, it analyzes each individual packet for anomalous features, such as the presence of certain strings, target ports, and inconsistent or dangerous headers. In contrast, our work utilizes comparisons across multiple server response messages and semantic properties derived from our data access and replication protocols. In general, it will not be feasible to perform this type of detection by examining individual network packets.

The goal of our work is substantially different from that of various recent works that have adapted *failure detectors* [CT96] to solve consensus in distributed systems that can suffer Byzantine failures [MR97, DS97, KMM97]. These works focus on the specification of abstract failure detectors that enable consensus to be solved. Our goal here is to develop techniques for detecting Byzantine server responses specifically in the context of data replicated using quorum systems, without regard to abstract failure detector specifications or the consensus problem.

Finally, Lin et al. [LRM98] analyze the process of gradual infection of a system by malicious entities. Their analysis attempts to project when failures exceed certain thresholds by extrapolating from observed failures onto the future, on the basis of certain a priori assumptions about the communication patterns of processes and the infection rate of the system. Our methods do not depend on these assumptions, as they do not address the propagation of failures in the system. Rather, they attempt to measure the current number of failures at any point in time.

3 Preliminaries

3.1 System model

Our system model is based on a *universe* U of n data servers. A *correct* server is one that behaves according to its specification, whereas a *faulty* server deviates from its specification arbitrarily (Byzantine failure) and, we assume, consistently. We denote the set of actually faulty servers by F, and express our failure assumption (equivalent to the *fail-prone set* of [MR98]) as follows:

Definition 1 *A* failure assumption $\mathcal{B} \subseteq 2^U$ *is a set of subsets of servers such that (i) if* $B_1 \in \mathcal{B}$ *and* $B_2 \subset B_1$*, then* $B_2 \in \mathcal{B}$*, and (ii) in any run of the system,* $F \in B$*.*

As an example of a failure assumption, consider the common assumption that there is some threshold t such that at most t servers fail. We call this the *t-threshold failure assumption*:

Definition 2 *The* t-threshold failure assumption *is* $\mathcal{B} = \{B \subseteq U \mid \#(B) \leq t\}$*.*

(Here and throughout this paper, we use $\#(S)$ to denote the cardinality of the set S.)

Our system model also includes some number of clients, which we assume to be correct. Clients communicate with servers over point-to-point channels. Channels are reliable, in the sense that a message sent between a client and a correct server is eventually received by its destination. In addition, a client can authenticate the channel to a correct server; i.e., if the client receives a message from a correct server, then that server actually sent it.

3.2 Masking quorum systems

In order to focus attention on our fault detection techniques, we consider a simple scenario in which each server holds a copy of some replicated variable Z, on which clients can execute *write* and *read* operations to change or observe its value, respectively. The protocols for writing and reading Z employ a *masking quorum system* [MR98, MRW99], or the probabilistic variations thereof [MRWW98].

Definition 3 *A* masking quorum system *for a failure assumption $\mathcal{B}$ is a set $\mathcal{Q} \subseteq 2^U$ of subsets of servers such that $\forall Q_1, Q_2 \in \mathcal{Q}$, $\forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$*

Intuitively, if each read and write is performed at a quorum of servers, then the use of a masking quorum system ensures that a read quorum Q_2 intersects the last write quorum Q_1 in a set $(Q_1 \cap Q_2) \setminus F$ such that this set can be distinguished as containing a correct server. This suffices to enable the reader to determine the last written value. A straightforward masking quorum system is the *Uniform masking quorum system* [MR98][1]:

Definition 4 *A* Uniform masking quorum system *for the t-threshold failure assumption is $\mathcal{Q} = \{Q \subset U \mid \#(Q) = \lceil \frac{n+2t+1}{2} \rceil\}$.*

A client's choice of quorum to access in any given protocol instance is determined by an *access strategy* w that is a probability distribution on the quorums: the client chooses

[1] This masking quorum system is named Threshold in [MR98]; we refer to it as Uniform to avoid confusion with the threshold failure assumption.

quorum Q with probability $w(Q)$. All probabilities from here on are computed when the choice of quorums is made according to the access strategy.

We consider the following protocols for accessing the replicated variable Z, which were shown in [MR98] to give Z the semantics of a *safe* variable [Lam86]. Each server u maintains a timestamp T_u with its copy Z_u of the variable Z. A client writes the timestamp when it writes the variable. These protocols require that different clients choose different timestamps, and thus each client c chooses its timestamps from some set $\mathcal{T}_c$ that does not intersect $\mathcal{T}_{c'}$ for any other client c'. Client operations proceed as follows.

Write: For a client c to write the value v to Z, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{ <Z_u, T_u>\}_{u \in Q}$, chooses a timestamp $T \in \mathcal{T}_c$ greater than the highest timestamp value in A and greater than any timestamp it has chosen in the past, and updates Z_u and T_u at each server u in some quorum Q' to v and T, respectively.

Read: For a client to read a variable Z, it queries each server in some quorum Q to obtain a set of value/timestamp pairs $A = \{<Z_u, T_u>\}_{u \in Q}$. From among all pairs returned by any subset $S \subseteq Q$ of servers that satisfies $\forall B \in \mathcal{B} : S \not\subseteq B$, the client chooses the pair $<v, T>$ with the highest timestamp T, and then returns v as the result of the read operation. If there is no such pair in A, the result of the read operation is $\perp$ (a null value).

In a write operation, each server u updates Z_u and T_u to the received values $<v, T>$ only if T is greater than the present value of T_u; this convention guarantees the serializability of concurrent writes. As mentioned in Section 1 we consider only reads that are not concurrent with writes. In this case, the read operation will never return $\perp$ (provided that the failure assumption is not violated).

3.3 Statistical building blocks

As described in Section 1, the primary goal of this paper is to detect when F is *dangerous*, i.e., there exists a set S of Δ servers such that $F \cup S \in \overline{\mathcal{B}}$, for some parameter Δ. To do this, we exploit information made available during the read protocol of Section 3.2, in conjunction with a basic statistical technique called *hypothesis testing*.

Hypothesis testing is based on the slightly counterintuitive concept of testing for a condition by looking for evidence *that its opposite is false*. More specifically, we formulate the condition to be tested for as an *experimental hypothesis* H_E (e.g., F is dangerous), and formulate its opposite as the complementary *null hypothesis* H_0 (e.g., F is not dangerous). We then perform an experiment (observe the value of a *random variable*) whose probability distribution differs under the two hypotheses, and watch for an outcome that supports H_E while being highly improbable under H_0 (we will discuss the exact meaning of "highly improbable" below). Such an outcome is considered evidence that H_E is true—in our context, that the set F of faulty servers in the system is dangerous.

The most obvious random variable for our purposes would be the set of faulty server

responses to a read request. For example, if a quorum is expected to have k faulty servers on average when F is not dangerous, and a read observed considerably more than k faulty responses, we would naturally regard this as evidence that F is dangerous. Unfortunately, the set of faulty servers in a read quorum is usually not directly observable (though we come close with our *write marker* approach).

Instead, each of our methods defines a random variable X whose value *can* be observed during a read operation, and whose probability distribution varies with F. As suggested above, we define H_E and H_0 as "F is dangerous" and "F is not dangerous", respectively. We treat the notion of "highly improbable" as a parameter of our detection algorithms by setting a *rejection level* α, $0 < \alpha < 1$. We then define a *region of rejection* for H_0 as a maximal range of values for X that have a combined probability under H_0 of at most α but that have a significantly greater probability under H_E. Any observed value of X that falls into this region of rejection is considered to be evidence that H_0 is false and H_E is true, i.e., that F is dangerous.

Note that the rejection level α must be chosen carefully: the higher it is, the greater the chances of false positives (i.e., alarms sent when F is not dangerous), but too low a value increases the chance of false negatives (failure to detect when F is dangerous). For reasonable values of α (e.g., $\alpha = 0.05$), however, we give examples showing that the former risk can be kept to a reasonable minimum, while the latter can be made essentially negligible.

In this paper we will typically choose $\Delta > 1$, as our primary goal is to detect dangerous conditions before the integrity of the data has been compromised. The "safest" value for Δ is the size of the smallest element of $\overline{\mathcal{B}}$ minus one, but a higher value may be desirable if

small numbers of faults are common and countermeasures are expensive.

4 Diagnosis using justifying sets

Our first method of fault detection for masking quorum systems uses the read and write protocols described in Section 3.2. As the random variable for our statistical analysis, we use the *justifying set* for a read operation, which is the set of servers that return the value/timestamp pair $<v, T>$ chosen by the client in the read operation. The justifying set can be as large as the read quorum if the read quorum is the same as the quorum used in the last completed write operation and contains no faulty servers, but may be significantly smaller if it contains faulty servers and intersects the last write quorum minimally.

Since a quantitative analysis of using the justifying set for fault detection is dependent on the failure assumption, we now narrow our attention to a particular example, namely the common t-threshold failure assumption (Defintion 2). In this case, F is dangerous if $\#(F) \geq t - \Delta$. For notational simplicity, we define $f = \#(F)$ and $t_a = t - \Delta$; t_a is called the *alarm line* of our tests. Specifically, suppose that a read operation is performed on the system, and that the size of the justifying set for that read operation is x. We would like to determine whether this evidence supports the hypothesis that F is dangerous, i.e., $f > t_a$. If for all failure scenarios F where $f \leq t_a$ the probability of observing a justifying set of size x is at most the rejection level α, then this suggests that the null hypothesis $f \leq t_a$ is false. That is, the region of rejection for the null hypothesis is defined as $x \leq$ highreject where highreject is the maximum value such that

$$\sum_{x=t+1}^{\mathsf{highreject}} P(\#((Q_1 \cap Q_2) \setminus F) = x) \leq \alpha \tag{1}$$

for all F such that $f \leq t_a$. Here, the probability $P(\#((Q_1 \cap Q_2) \setminus F) = x)$ is taken with respect to choices of the last write quorum Q_1 and the read quorum Q_2, according to the access strategy w for choosing quorums.

Computing the region of rejection depends on the particular quorum system and strategy in use. Here we demonstrate this for size-based quorum constructions, where a quorum consists of q servers chosen uniformly at random from a universe of n servers. This template includes both the Uniform construction of Definition 4, where $q = \lceil \frac{n+2t+1}{2} \rceil$, and the probabilistic construction of [MRWW98], where $q = \ell t$ and $2 \leq \ell \leq n/t$. For such quorum systems, rather than computing highreject directly using (1), it is simpler to compute highreject conservatively to be the maximum value such that

$$\sum_{\hat{f}=0}^{t_a} \sum_{x=t+1}^{\mathsf{highreject}} P(\#((Q_1 \cap Q_2) \setminus F) = x | \#(F) = \hat{f}) \leq \alpha \tag{2}$$

since $P(\#((Q_1 \cap Q_2) \setminus F) = x | \#(F) = \hat{f})$ is the same for any failure scenario F with $\hat{f}$ failures. Note that

$$\max_{F:\#(F)\leq t_a} \sum_{x=t+1}^{\mathsf{highreject}} P(\#((Q_1 \cap Q_2) \setminus F) = x) \;\leq\; \sum_{\hat{f}=0}^{t_a} \sum_{x=t+1}^{\mathsf{highreject}} P(\#((Q_1 \cap Q_2) \setminus F) = x | \#(F) = \hat{f})$$

for any value of highreject. So, computing highreject according to (2) gives a potentially smaller region of rejection than when computed according to (1), i.e., one that will raise an alarm less frequently. In the remainder of this section, we complete the computation

of highreject according to (2), and show that nevertheless the computed region effectively detects the case when f passes t_a.

Given $\hat{f}$ faulty servers in the system, the probability of exactly j failures in the read quorum can be expressed by a *hypergeometric distribution* as follows:

$$\frac{\binom{\hat{f}}{j}\binom{n-\hat{f}}{q-j}}{\binom{n}{q}}$$

Given that the number of failures in the read quorum is j, the probability that there are exactly x correct servers in the intersection between the read quorum and the previous write quorum is formulated as follows: the number of ways of choosing x correct servers from the read quorum is $\binom{q-j}{x}$, and the number of possible previous write quorums that intersect the read quorum in exactly those correct servers (and some number of incorrect ones) is $\binom{n-q+j}{q-x}$. The probability that the previous write quorum intersects the read quorum in exactly this way is therefore:

$$\frac{\binom{q-j}{x}\binom{n-q+j}{q-x}}{\binom{n}{q}}$$

To get the overall probability that there are exactly x correct servers in the intersection between the read and most recent write quorums, i.e., that the justifying set size is x, we multiply the conditional probability given j failures in the read quorum by the probability of exactly j failures in the read quorum, and sum the result for $j = 0$ to $\hat{f}$:

$$P(\#((Q_1 \cap Q_2) \setminus F) = x | \#(F) = \hat{f}) = \sum_{j=0}^{\hat{f}} \frac{\binom{q-j}{x}\binom{n-q+j}{q-x}\binom{\hat{f}}{j}\binom{n-\hat{f}}{q-j}}{\binom{n}{q}^2} \tag{3}$$

This formula expresses the probability that a particular read operation in a t-masking quorum system will have a justifying set size of x given the presence of $\hat{f}$ faults.

For a given rejection level α, then, the region of rejection for the null hypothesis $f \leq t_a$ is defined as $x \leq \mathsf{highreject}$, where $\mathsf{highreject}$ is the maximum value such that:

$$\sum_{\hat{f}=0}^{t_a} \sum_{x=t+1}^{\mathsf{highreject}} \sum_{j=0}^{\hat{f}} \frac{\binom{q-j}{x}\binom{n-q+j}{q-x}\binom{\hat{f}}{j}\binom{n-\hat{f}}{q-j}}{\binom{n}{q}^2} \leq \alpha$$

The left-hand expression above represents the *significance level* of the test, i.e., the probability of a false positive (false alarm).

If there are in fact $f > t_a$ failures in the system, the probability of detecting this condition on a single read is:

$$\sum_{x=t+1}^{\mathsf{highreject}} \sum_{j=0}^{f} \frac{\binom{q-j}{x}\binom{n-q+j}{q-x}\binom{f}{j}\binom{n-f}{q-j}}{\binom{n}{q}^2}$$

If we denote this value by γ, then the probability that k consecutive reads *fail* to detect the condition is $(1-\gamma)^k$. As shown in the following examples, k need not be very large for this probability to become negligible.

Example 1: Consider a threshold masking quorum system of $n = 101$ servers, a quorum size $q = 76$, and a fault tolerance threshold $t = 25$. In order to test whether there are *any* faults in the system, we set $t_a = 0$, so that the null hypothesis H_0 is $f = 0$ and the experimental hypothesis H_E is $f > 0$. Plugging these numbers into (3) over the full range of x yields the results in Table 1. For all other values of x not shown in Table 1, the probability of a justifying set of size x given $f = 0$ is zero.

x	$P(\#((Q_1 \cap Q_2) \setminus F) = x \mid f = 0)$	x	$P(\#((Q_1 \cap Q_2) \setminus F) = x \mid f = 0)$
51	.000243	64	0.000500
52	.002922	65	7.92×10^{-05}
53	.015880	66	9.68×10^{-06}
54	.051857	67	9.03×10^{-07}
55	.114087	68	6.33×10^{-08}
56	.179687	69	3.26×10^{-09}
57	.210160	70	1.20×10^{-10}
58	.186867	71	3.05×10^{-12}
59	.128273	72	5.03×10^{-14}
60	.068649	73	5.02×10^{-16}
61	.028810	74	2.65×10^{-18}
62	.009504	75	5.89×10^{-21}
63	.002464	76	3.10×10^{-24}

Table 1: Probability distribution on justifying set sizes for Example 1

Since $P(51)+P(52)+P(53) \approx 0.019$, while $P(51)+P(52)+P(53)+P(54) \approx 0.071$, the region of rejection for $\alpha = 0.05$ is defined as $x \leq 53$; if a read operation has a justifying set of size 53 or less, the client rejects the null hypothesis and concludes that there are faults in the system. This test has a *significance level* of 0.019; that is, there is a probability of 0.019 that the client will detect faults when there are none. (If this level of risk is unacceptable for a particular system, α can be set to a lower value, thus creating a smaller region of rejection.)

Suppose that there are actually f failures in the system. The probability that this experiment will detect the presence of failures during any given read is:

$$\sum_{x=26}^{53} \sum_{j=0}^{f} \frac{\binom{76-j}{x}\binom{25+j}{76-x}\binom{f}{j}\binom{101-f}{76-j}}{\binom{101}{76}^2}$$

Figure 1 shows these values for $1 \leq f \leq 25$.

Although the probability of detecting faults during a given read in this system is relatively low for very small values of f, it would appear that this test is reasonably powerful. Even for fault levels as low as 4 or 5, a client can reasonably expect to detect the presence of failures within a few reads; e.g., if $f = 5$, then the probability of detecting that $f > t_a$ in only six reads is already $1 - (1 - .345534)^6 = .921$. As the fault levels rise, the probability of such detection within a single read approaches near-certainty.

Example 2: Consider a much smaller threshold system consisting of $n = 61$ servers. The maximum tolerance threshold t for such a system is 15, with a quorum size $q = 46$. Furthermore, suppose that the administrator of this system does not wish to be notified if fewer than 5 failures occur, and therefore sets t_a to 5 rather than 0. In this situation, the

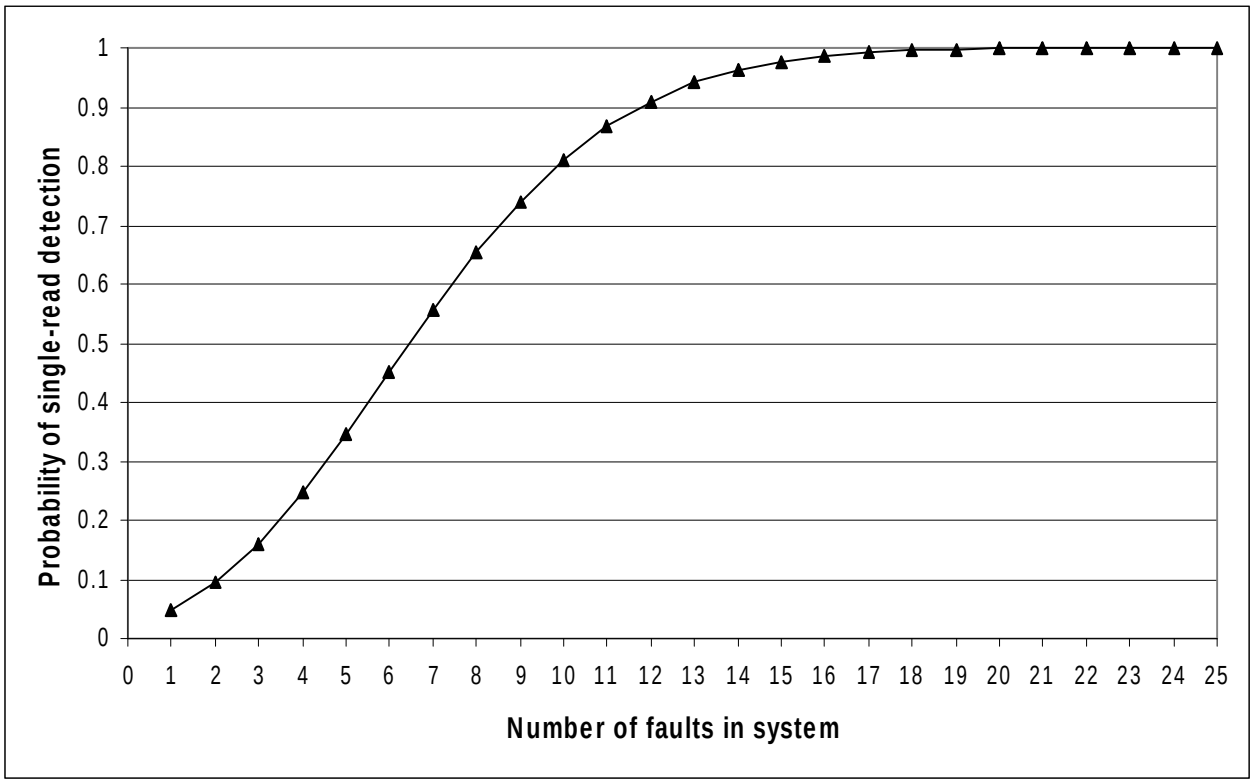

Figure 1: Probability of single-read detection for $n = 101, t_a = 0$ (Example 1)

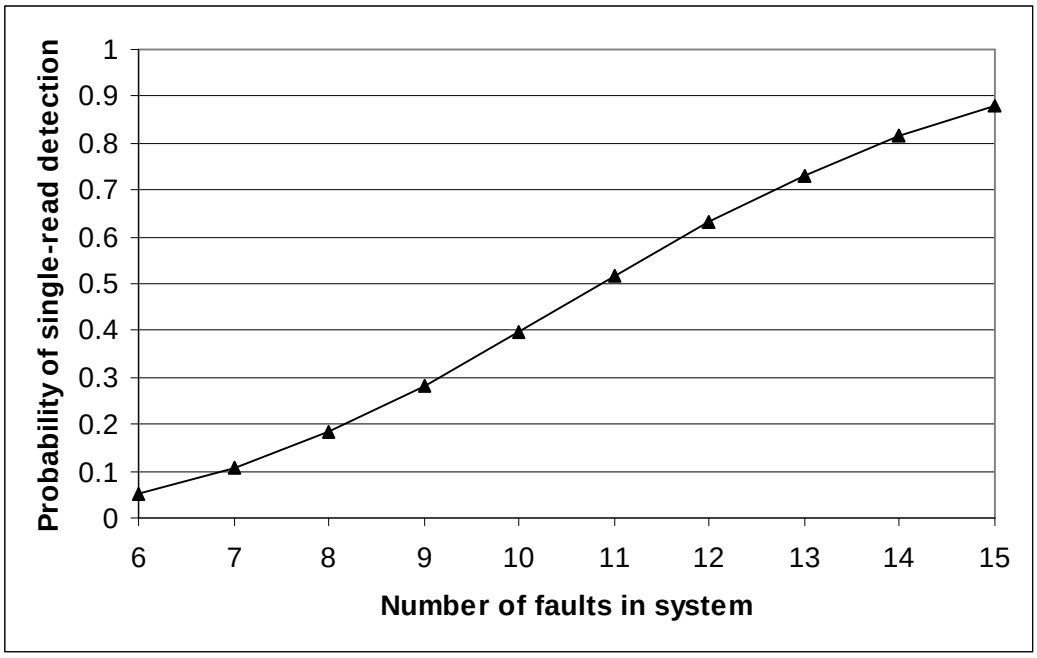

Figure 2: Probability of single-read detection for $n = 61, t_a = 5$ (Example 2)

null hypothesis H_0 is $f \leq t_a$. Given $\alpha = 0.05$, the region of rejection for H_0 can be computed to be $x \leq 28$ by similar means to those used in Example 1. The probabilities of detecting this condition for actual values of f between 6 and 15 inclusive are shown in Figure 2.

As one might expect, error conditions are more difficult to detect when they are more narrowly defined (e.g., $5 < f < 15$ vs. $0 < f < 25$), as the contrast between Examples 1 and 2 shows. Even in the latter experiment, however, a client can reasonably expect to detect a serious but non-fatal error condition within a small number of reads. For $f = 12$,

the probability that the alarm is triggered within six read operations is $1 - (1 - 0.428527)^6$, approximately .965. The probability that it is triggered within ten reads is over .996. We can therefore reasonably consider this technique to be a useful diagnostic in systems where read operations are significantly more frequent than server failures, particularly if the systems are relatively large.

A similar analysis could be performed for masking quorum systems for a t-threshold failure assumption other than Uniform masking systems, taking (1) as the goal to compute. Rather than focusing on another example of this computation, however, we instead move on to a more powerful method of fault detection. The technique of the present section gives little indication of the specific number of faults that have occurred and provides little information toward identifying which servers are faulty. Our next diagnostic method addresses both these needs.

5 Diagnosis using quorum markers

The diagnostic method presented in this section has two distinct functions. First, it estimates the fault distribution over the whole system using a technique similar to that of the previous section, but with greater precision. Second, it pinpoints specific servers that exhibit detectably faulty behavior during a given read. The diagnostic operates on an enhanced version of the read/write protocol for masking quorum systems: the write marker protocol, described next.

5.1 The write marker protocol

The *write marker protocol* uses a simple enhancement to the read/write protocol of Section 3.2. For a replicated variable Z, there is an accompanying replicated variable $\underline{Z}$ that stores an identifier for the quorum (e.g., an n-bit vector indicating the servers in the quorum) that was used to complete the write operation in which Z was last written. The write protocol proceeds as in Section 3.2, except that after the write to Z_u at each $u \in Q'$ completes, the client writes (the identifier for) Q' to $\underline{Z}_u$. We reiterate that for the purposes of this paper, we assume that all of this occurs without interference by a different client's concurrent operation.

The read protocol proceeds essentially as before, except that each server returns the triple $<Z_u, T_u, \underline{Z}_u>$ in response to a read request, where $\underline{Z}_u$ is u's replica of $\underline{Z}$. From among all triples returned from any set S of servers where $S \not\subseteq B$ for all $B \in \mathcal{B}$, the client chooses the triple with the highest timestamp.

We now describe how to use the triples returned by the servers to monitor fault levels in various types of quorum systems. At the conclusion of this section we also show how they can be used to specifically identify a subset of the faulty servers.

5.2 Fault detection

Our revised statistical technique uses the quorum markers to determine the set of servers whose returned values are expected to match the accepted triple in the absence of faults (i.e., the set of servers in the intersection between the read quorum and the previous write

quorum), and the subset of those servers whose returned values actually do match that triple. With this extra information, we can substantially refine the technique of Section 4 to yield greater accuracy. Specifically, we can use this information to determine the probability of having observed a justifying set given a failure scenario F *and given that the intersection of the read and previous write quorums is a known set* S. For example, consider again the t-threshold failure assumption. Then, following our reasoning from Section 4, the region of rejection for the null hypothesis is defined as $x \leq \mathsf{highreject}$ where $\mathsf{highreject}$ is the maximum value such that

$$\sum_{x=t+1}^{\mathsf{highreject}} P(\#((Q_1 \cap Q_2) \setminus F) = x \mid Q_1 \cap Q_2 = S) \leq \alpha \tag{4}$$

for all F such that $\#(F) \leq t_a$. Again, the probability $P(\#((Q_1 \cap Q_2) \setminus F) = x \mid Q_1 \cap Q_2 = S)$ here is computed with respect to choices of Q_1 and Q_2 according to the strategy w for choosing quorums (and satisfying $Q_1 \cap Q_2 = S$). Computing the region of rejection depends on the quorum system and access strategy in use.

5.2.1 Size-based quorum systems

To illustrate computing a region of rejection when the write marker protocol is used, we return to size-based quorum systems in which a quorum is defined as a selection of q servers uniformly at random. Because of the random selection of the servers that make up the quorum for a given operation, $P(\#((Q_1 \cap Q_2) \setminus F) = x \mid \#(F) = \hat{f}, Q_1 \cap Q_2 = S)$ is the same for any F and any S of size $s = \#(S)$. Specifically, this can be expressed by the

hypergeometric formula:

$$P(\#((Q_1 \cap Q_2) \setminus F) = x \mid \#(F) = \hat{f}, \#(Q_1 \cap Q_2) = s) = \frac{\binom{\hat{f}}{s-x}\binom{n-\hat{f}}{x}}{\binom{n}{s}} \ .$$

Following the treatment in Section 4 for size-based quorums, we compute a region of rejection for the null hypothesis as the highest value $\mathsf{highreject} \leq s$ such that

$$\sum_{\hat{f}=0}^{t_a} \sum_{x=t+1}^{\mathsf{highreject}} \frac{\binom{\hat{f}}{s-x}\binom{n-\hat{f}}{x}}{\binom{n}{s}} \leq \alpha$$

Again, the left-hand expression represents the probability of a false alarm.

The increased strength of this method turns experiments in which $t_a = 0$ into a degenerate case. The presence of *any* faults in the intersection set is visible and invalidates the null hypothesis; the probability of a false positive in such cases is zero, as the formula above confirms. Likewise, as the number of faults increases, the probability of detecting faults within one or two reads rapidly approaches certainty.

Example 3: Consider again the system of $n = 101$ servers, with a fault tolerance threshold of $t = 25$, a quorum size of $q = 76$, and $t_a = 0$, and suppose that a given read quorum overlaps the previous write quorum in $s = 57$ servers (the most likely overlap, with a probability of about 0.21). Here, the only possible justifying set size under the assumption $f = 0$ is $x = s$, and hence, it is rejected for any $x < s$. The probability of alarm on a single read operation for various values of $f < t$ is shown in Figure 3, which also illustrates the dramatically higher precision of the write-marker method over the justifying set method.

The added precision of the write marker method has additional advantages when t_a is set to a value greater than zero, as shown in the next example.

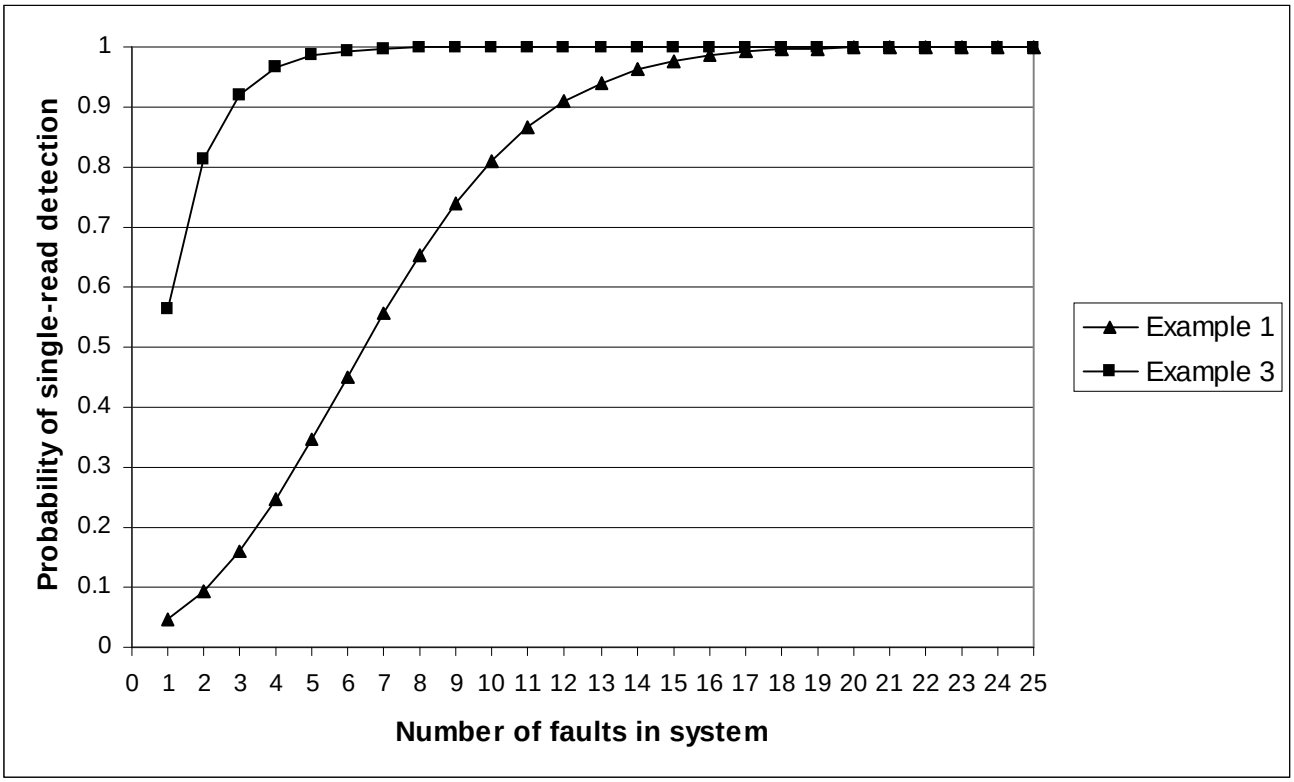

Figure 3: Comparison of Examples 1 and 3

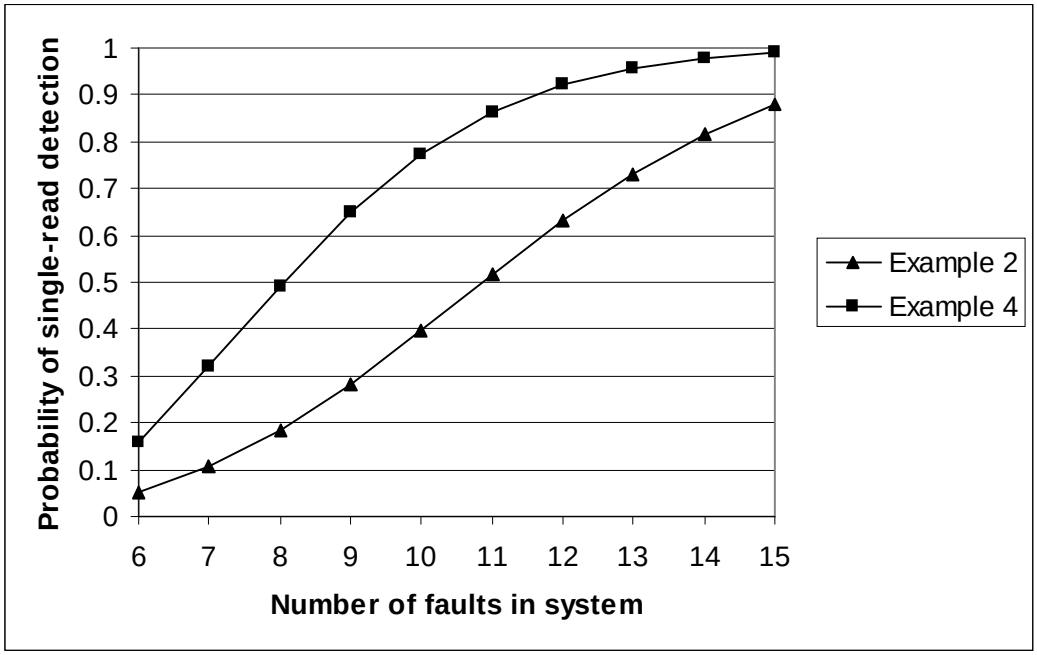

Figure 4: Comparison of Examples 2 and 4

Example 4: Consider again the system of $n = 61$ servers, with a fault tolerance threshold of $t = 15$, a quorum size of $q = 46$, and $t_a = 5$, and suppose that a given read quorum overlaps the previous write quorum in the common intersection size $s = 34$ servers. The region of rejection for the null hypothesis $f \leq 5$, calculated using the formula above, is $x \leq 29$. The probability of alarm on a single read operation for various values of f, $t_a < f < t$, is shown in Figure 4. Again, the increased strength of the write-marker method is evident.

Like the method presented in Section 4, the write-marker technique has the advantage of flexibility. If we wish to minimize the risk of premature alarms (i.e., alarms that are

sent without the alarm threshold being exceeded) we may choose a smaller α at the risk of somewhat delayed alarms. In fact, the greater precision of this method decreases the risks associated with such a course: even delayed alarms can be expected to be timely.

A Note on Smaller Systems

Our discussion so far has centered around large systems. For the technique described in Section 4 this restriction is necessary, as the method is not powerful enough to detect faults reliably in small systems. As the comparisons above show, however, the write marker method is considerably stronger, and is in fact suitable for fault detection in more moderately-sized threshold systems, even when the alarm line is greater than zero.

Example 5: Consider a system where $n = 25$, $t = 6$, $q = 19$, and $t_a = 2$. Suppose a read quorum overlaps a write quorum in the most common intersection size of 14. The probability of detecting $f > 2$ on a single read is moderately high; the probability of detecting the condition within k reads increases dramatically with k, as shown in Figure 5.

5.2.2 Quorum systems for other failure assumptions

Fault detection using the write marker technique can be applied for failure assumptions other than the threshold one. In this section, we demonstrate this using a more sophisticated type of quorum system: the BoostFPP system.

A BoostFPP masking quorum system is described in [MRW99]. It is constructed as a composition of two quorum systems. The first is a quorum system based on a finite

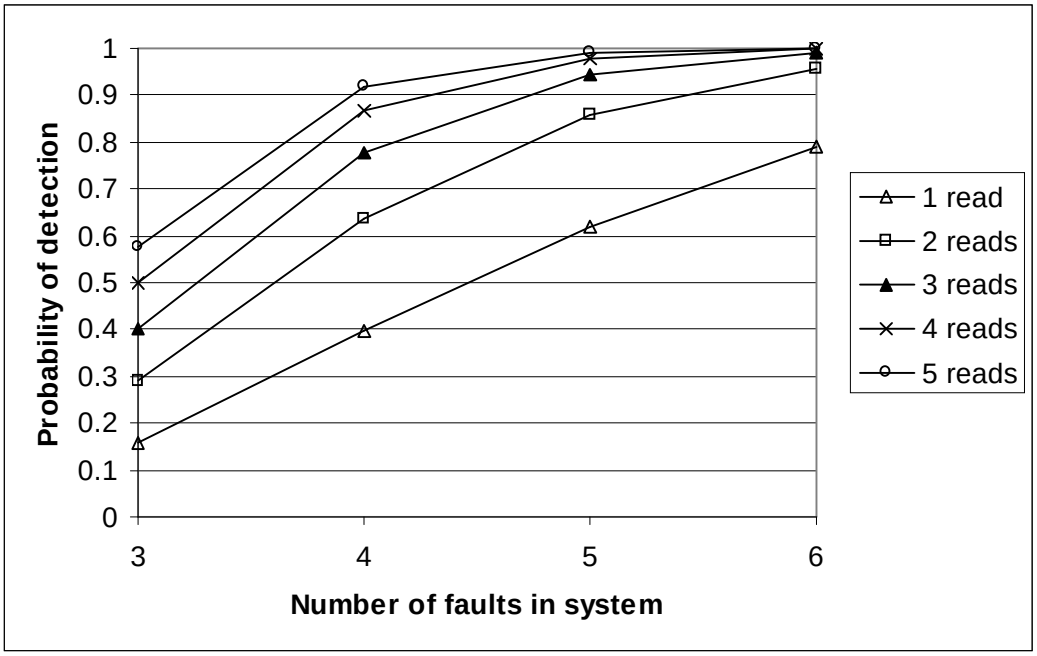

Figure 5: Probability of detection within 1–5 reads for $n = 25, t_a = 2$ (Example 5)

projective plane (FPP), suggested originally as a quorum system by [Mae85]. In the FPP quorum system, there are $q^2 + q + 1$ elements and quorums of size $q + 1$ (corresponding to the hyperplanes of the FPP), where $q = p^r \geq 2$ for some prime p and integer r. Each pair of distinct quorums in FPP intersect in exactly one element. For the second quorum system we again employ the Uniform masking quorum system [MR98] with a universe of size $4t + 1$ and quorums of size $3t + 1$. The composition of the two systems is made by replacing each element of the FPP with a distinct copy of a threshold system. That is, the universe for a BoostFPP system is $U = \bigcup_{i=1}^{q^2+q+1} U_i$, where each U_i is a set of $4t + 1$ servers, and $U_i \cap U_j = \emptyset$ for any $i \neq j$. We refer to each U_i as a "super element". A quorum is chosen by first selecting a quorum of super elements in the FPP, say $U_{i_1}, \ldots, U_{i_{q+1}}$, and then selecting $3t + 1$ servers from each U_{i_j}. Figure 6 depicts a BoostFPP system with $q = 2$ and $t = 1$, with one quorum shaded. We adopt a uniform access strategy, assigning equal access probability to every quorum.

We use the following notation to "map" between super elements and servers and vice versa:

1. If X is a set of servers, then $\mathsf{super}(X)$ is the set of super elements such that $U_i \in \mathsf{super}(X)$ iff $U_i \cap X \neq \emptyset$.

2. If Y is a set of servers and U_i is a super-element, then $[Y]_{U_i} = Y \cap U_i$.

BoostFPP can tolerate the failure of up to t servers in each one of its super elements. Hence, the failure assumption it masks is

$$\mathcal{B} = \{B \subseteq U \mid \forall i : (B \cap U_i) \leq t\} \ .$$

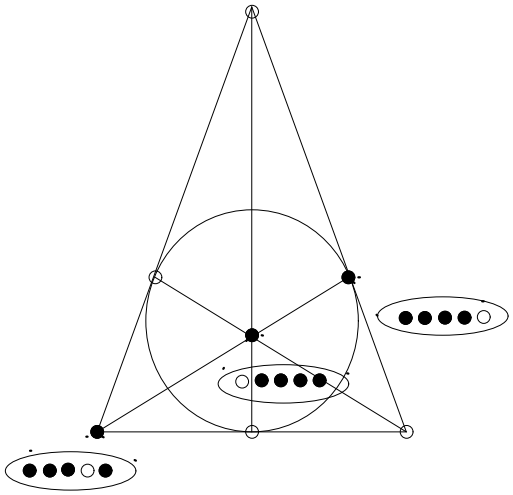

Figure 6: BoostFPP with $q = 2$ and $t = 1$, with one quorum shaded

We now apply our techniques, using the write marker protocol of Section 5, to detect when F is dangerous. In this case,

$$\overline{\mathcal{B}} = \{B \subseteq U \mid \exists i : (B \cap U_i) > t\} \ .$$

Therefore, we set a threshold $t_a = t - \Delta$ and aim to detect if **any** super element contains more than t_a faulty servers. More precisely, we define a region of rejection as $x \leq \mathsf{highreject}$ where $\mathsf{highreject}$ is the maximum value such that

$$\sum_{U_i \in \mathsf{super}(Q_1 \cap Q_2)} \sum_{x=t+1}^{\mathsf{highreject}} P(\#(([Q_1]_{U_i} \cap [Q_2]_{U_i}) \setminus [F]_{U_i}) = x \mid \#([Q_1]_{U_i} \cap [Q_2]_{U_i}) = s_i) \ \leq \ \alpha$$

Borrowing from the analysis of Section 5.2.1, this becomes

$$\sum_{U_i \in \mathsf{super}(Q_1 \cap Q_2)} \sum_{\hat{f}=0}^{t_a} \sum_{x=t+1}^{\mathsf{highreject}} \frac{\binom{\hat{f}}{s_i - x}\binom{4t+1-\hat{f}}{x}}{\binom{4t+1}{s_i}} \ \leq \ \alpha$$

Recall that our aim is to detect when the number of failures in the system approaches a failure scenario in $\overline{\mathcal{B}}$, i.e., when in some super element the number of faulty servers exceeds t_a. Since the choice of quorums randomizes the choice of super elements that fall in the intersection of read and write quorums, the likelihood of detecting a dangerous condition within several reads is derived from the probability of detecting the corresponding dangerous condition within any super element.

5.3 Fault identification

The write marker protocol has an even stronger potential as a tool for fault detection: it allows the client to identify specific servers that are behaving incorrectly. By keeping a record of this list, the client can thereafter select quorums that do not contain these servers. This allows the system to behave more efficiently than it would otherwise, as well as to gather the information needed to isolate faulty servers for repair to preserve the system's integrity.

The fault identification algorithm accepts as input the triples $\{<Z_u, T_u, \underline{Z}_u>\}_{u \in Q}$ that the client obtained from servers in the read protocol, as well as the triple $<v, T, Q'>$ that the client chose as the result of the read operation. It then computes the set $S \setminus S'$ where $S = Q \cap Q'$ and S' is the set of servers that returned $<v, T, Q'>$ in the read operation. The servers in $S \setminus S'$ are identified as faulty.

Note that the fault identification protocol does not depend in any way on the specific characteristics of the quorum system, and can be applied to masking quorum systems in general.

6 Choosing alarm lines for large systems

The analysis of the previous two sections is precise but computationally cumbersome for very large systems. A useful alternative is to estimate the performance of possible alarm lines by means of bound analysis. In this section we present an asymptotic analysis of the techniques of Sections 4 and 5 that shows how to choose an alarm line value for arbitrarily large systems. Here we return our focus to the t-threshold failure assumption and size-based quorum constructions with quorum size q.

Let Q denote a read quorum, Q' a write quorum, F the set of faulty servers, and $t_a = t - \Delta$ the alarm line. For every $\hat{f} \leq t_a$, we define a random variable $X_{\hat{f}} = \#((Q \cap Q') \setminus F)$, where $\#(F) = \hat{f}$, and $\#((Q \cap Q') \setminus F)$ is the justifying set size. We can compute the expectation of $X_{\hat{f}}$ directly. For each server $u \notin F$ define an indicator random variable I_u such that $I_u = 1$ if $u \in (Q \cap Q') \setminus F$ and $I_u = 0$ otherwise. For such u we have $P(I_u = 1) = q^2/n^2$ since Q and Q' are chosen independently. By linearity of expectation,

$$E[X_{\hat{f}}] = \sum_{u \in U \setminus F} E[I_u] = \sum_{u \in U \setminus F} P(I_u = 1) = (n - \hat{f})\frac{q^2}{n^2}.$$

Intuitively, the distribution on $X_{\hat{f}}$ is centered around its expectation and decreases exponentially as $X_{\hat{f}}$ moves farther away from that expectation. Thus, we should be able to show that $X_{\hat{f}}$ grows smaller than its expectation with exponentially decreasing probability. A tempting approach to analyzing this would be to use Chernoff bounds, but these do not directly apply because the selection of individual servers in Q (similarly, Q') is not independent. In the analysis below, we thus use a more powerful tool, *martingales*, to derive the anticipated Chernoff-like bound.

6.1 Martingales

We provide here a brief introduction to martingales, which summarizes only the necessary definitions and results from the more thorough treatment found in [MR95, Ch. 4.4].

Definition 5 *A* martingale *sequence is a sequence of random variables* $X_0, X_1, \ldots$ *such that for all* $i > 0$,

$$E[X_i \mid X_0, \ldots, X_{i-1}] = X_{i-1}$$

Our goal in constructing martingale sequences is to apply the following theorem.

Theorem 1 (Azuma's Inequality, [MR95]) *Let* $X_0, X_1, \ldots$ *be a martingale sequence such that for each* k,

$$\#(X_k - X_{k-1}) \leq c$$

where c *is independent of* k. *Then, for* $t \geq 0$ *and* $\delta > 0$,

$$P(\#(X_t - X_0) \geq \delta) \leq 2e^{-\frac{\delta^2}{2tc^2}}$$

The particular method that we use for constructing martingale sequences employs the notion of a *filter* over a finite sample space Ω, which is a nested sequence of event-sets $F_0 \subseteq F_1 \subseteq \cdots \subseteq F_k$ where $F_0 = \{\emptyset, \Omega\}$, $F_k = 2^\Omega$, and for $0 \leq i \leq k$, F_i is closed under complement and union. Intuitively, each F_i can be thought of as being generated by a partition of Ω into disjoint events, where F_{i+1} is generated by a more refined partition than F_i. In Section 6, each block (event) of the partition generating F_i is defined by the first i choices of servers in each of two quorums. We then apply the following theorem to construct a Doob martingale:

Theorem 2 (Doob martingale, [MR95]) *Let $F_0, \ldots, F_k$ be a filter, let X be any random variable, and define $X_i = E[X \mid F_i]$, i.e., X_i is the expected value of X conditioned on the events in F_i. Then $X_0, \ldots, X_k$ is a martingale.*

6.2 Deriving the bound

We bound the probability $P(X_{\hat{f}} < k)$ using the method of bounded differences, by defining a suitable Doob martingale sequence and applying Azuma's inequality. Here, a Doob martingale sequence of conditional random variables is defined by setting $X_{\hat{f},i}$, $0 \leq i \leq q$, to be the expected value of $X_{\hat{f}}$ after i selections are made in each of Q and Q'. Then, $X_{\hat{f}} = X_{\hat{f},q}$ and $E[X_{\hat{f}}] = X_{\hat{f},0}$, and it is not difficult to see that $\#(X_{\hat{f},i} - X_{\hat{f},i-1}) \leq 2$ for all $1 \leq i \leq q$. This yields the following bound:

$$P(X_{\hat{f}} < E[X_{\hat{f}}] - \delta) \leq 2e^{-\frac{\delta^2}{8q}}$$

Moreover, this bound is largest for $\hat{f} = t_a$. This is because if $\hat{f} \leq t_a$, then $E[X_{\hat{f}}] \geq E[X_{t_a}]$, and so $E[X_{\hat{f}}] - \delta' = E[X_{t_a}] - \delta$ implies $\delta' \geq \delta$, and the inequality on the corresponding bounds follows. Hence, we use this formula and our desired rejection level α to determine a δ such that $P(X_{t_a} < E[X_{t_a}] - \delta) \leq \alpha$. Thus we bound our probability of a false alarm and can diminish it by decreasing α and recalculating δ. The value $E[X_{t_a}] - \delta$ defines our region of rejection (see Section 3.3).

In order to analyze the probability that our alarm is triggered when the number of faults in the system is $f > t_a$, we use the same analysis to obtain:

$$E[X_f] = (n - f)\frac{q^2}{n^2} < (n - t_a)\frac{q^2}{n^2} = E[X_{t_a}]$$

An analysis similar to the above provides the following bound:

$$P(X_f > E[X_f] + \delta') \leq 2e^{-\frac{\delta'^2}{8q}}$$

To summarize, these bounds can now be used as follows. For any given alarm line t_a, and any desired confidence level α, we can compute the minimum δ to satisfy $2e^{-\frac{\delta^2}{8q}} \leq \alpha$. We thus derive the following test: An alarm is triggered whenever the justifying set size is less than $(n - t_a)\frac{q^2}{n^2} - \delta$. The analysis above guarantees that this alarm will be triggered with false positive probability at most our computed bound $2e^{-\frac{\delta^2}{8q}} \leq \alpha$. If, in fact, f faults occur and f is sufficiently larger than t_a, then there exists $\delta' > 0$ such that $E[X_f] + \delta' = E[X_{t_a}] - \delta$. Then, by the analysis above, the probability of triggering the alarm is greater than $1 - 2e^{-\frac{\delta'^2}{8q}}$.

In the case of the write marker protocol, we can tighten the analysis by using the (known) intersection size between Q and Q' as follows. Define $S = Q \cap Q'$, $s = \#(S)$, and a random variable $Y = \#(S \setminus F)$. Y has a hypergeometric distribution on s, $n - t_a$, and n, and $E[Y] = s(n - t_a)/n$. The appropriate Doob martingale sequence in this case defines Y_i, $0 \leq i \leq s$, to be the expected value of Y after i selections are made in S. Then, $\#(Y_i - Y_{i-1}) \leq 1$, and so to set the region of rejection we can use

$$P(Y < E[Y] - \delta) \leq 2e^{-\frac{\delta^2}{2s}}.$$

7 Conclusion

In this paper, we have presented two methods for probabilistic fault diagnosis for services replicated using masking quorum systems. Our methods mine server responses to read oper-

ations for evidence of server failures, and if necessary trigger an alarm to initiate appropriate recovery actions. We demonstrated both of our methods in the context of size-based constructions in which quorums are chosen of size q uniformly at random. We additionally demonstrated the write marker technique for a non-uniform construction, that of the Boost-FPP quorum system. Our first method has the advantage of requiring no modifications to the read and write protocols proposed in [MR98]. The second method requires minor modifications to these protocols, but offers better diagnosis capabilities and a precise identification of faulty servers. Our methods are very effective in detecting faulty servers, since faulty servers risk detection in every read operation for which they return incorrect answers.

A possible direction for future direction is fault detection using an aggregate of responses from multiple read queries. This would potentially allow us to eliminate some of the restricting simplifications we made, particularly that reads do not overlap any writes.

References

[AMPR99] L. Alvisi, D. Malkhi, E. Pierce and M. Reiter. Fault detection for Byzantine quorum systems. In *Proceedings of the 7th IFIP International Working Conference on Dependable Computing for Critical Applications*, pages 357–371, January 1999.

[Amo99] E. Amoroso. *Intrusion Detection: An Introduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response.* Intrusion.Net Books, 1999.

[Baz97] R. Bazzi. Synchronous Byzantine quorum systems. In *Proceedings of the 16th ACM Symposium on Principles of Distributed Computing*, pages 259–266, 1997.

[BB93] R. W. Buskens and R. P. Bianchini, Jr. Distributed on-line diagnosis in the presence of arbitrary faults. In *Proceedings of the 23rd International Symposium on Fault-Tolerant Computing*, pages 470–479, June 1993.

[CT96] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[CMR00] G. Chokler, D. Malkhi and M. Reiter. Backoff Protocols for Distributed Mutual Exclusion and Ordering. Submitted for publication.

[HC81] S. L. Hakimi and K.-Y. Chwa. Schemes for fault-tolerant computing: a comparison of modularly redundant and t-diagnosable systems. *Inform. Contr.*, 49:212–238, June 1981.

[DS97] A. Doudou and A. Schiper. Muteness detectors for consensus with Byzantine processes. Technical Report TR97-230, Department of Computer Science, École Polytechnic Fédérale de Lausanne, October 1997.

[KMM97] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. Solving consensus in a Byzantine environment using an unreliable failure detector. In *Proceedings of the International Conference on Principles of Distributed Systems*, pages 61–75, December 1997.

[Lam86] L. Lamport. On interprocess communication (part II: algorithms). *Distributed Computing* 1:86–101, 1986.

[LRM98] M. J. Lin, A. Ricciardi and K. Marzullo. On the resilience of multicasting strategies in a failure-propagating environment. UT Austin TR-1998-003.

[Mae85] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems* 3(2):145–159, 1985.

[MM81] J. Maeng and M. Malek. A comparison connection assignment for self-diagnosis of multiprocessor systems. In *Proceedings of the 11th International Symposium on Fault-Tolerant Computing*, pages 173–175, 1981.

[Mal80] M. Malek. A comparison connection assignment for diagnosis of multiprocessor systems. In *Proceedings of the 7th International Symposium on Computer Architecture*, pages 31–35, 1980.

[MR99] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*. To appear.

[MR98] D. Malkhi and M. K. Reiter. Byzantine quorum systems. *Distributed Computing* 11(4):203–213, 1998.

[MR97] D. Malkhi and M. K. Reiter. Unreliable intrusion detection in distributed computation. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop*, pages 116–124, June 1997.

[MRW99] D. Malkhi, M. K. Reiter, and A. Wool. The load and availability of Byzantine quorum systems. *SIAM Journal of Computing*. To appear.

[MRWW98] D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic Byzantine quorum systems. Brief announcement in *Proceedings of the 17th ACM Symposium on Principles of Distributed Computing*, page 321, June 1998. Full version submitted for publication.

[MR95] R. Motwani and P. Raghavan. *Randomized algorithms*. Cambridge University Press, 1995.

[SR87] K. Shin and P. Ramanathan. Diagnosis of processors with Byzantine faults in a distributed computing system. In *Proceedings of the 17th International Symposium on Fault Tolerant Computing*, pages 55–60, 1987.

Lorenzo Alvisi is an Assistant Professor and Faculty Fellow in the Department of Computer Sciences at the University of Texas at Austin. His primary research interests are in reliable distributed computing. He holds M.S. (1994) and Ph.D. (1996) degrees in Computer Science from Cornell University and a Laurea *summa cum laude* in Physics (1987) from

the University of Bologna, Italy. Dr. Alvisi is the recipient of an Alfred P. Sloan Research Fellowship, an IBM Faculty Partnership award, and an NSF CAREER award.

Dahlia Malkhi is a faculty member of the School of Computer Science and Engineering at the Hebrew University of Jerusalem, Jerusalem, Israel, where she heads the Secure Systems Research Laboratory. She received her Ph.D., M.Sc. degrees in computer science and a B.Sc. degree in mathematics and computer science in 1994, 1988, 1985, respectively, from the Hebrew University of Jerusalem, Israel. She was a member of the Secure Systems Research Department at AT&T Labs-Research in Florham Park, New Jersey from 1995 to 1999. Her research interests include all areas of distributed systems and security.

Evelyn Pierce is a recent graduate of the University of Texas at Austin, where she received her Ph.D. in Computer Sciences in December, 2000. She received her M.S. in Computer Sciences from the University of Texas at Austin in December 1993, and her B.A. in Philosophy from Princeton University in 1986. Her primary research interests are fault tolerance and security for asynchronous distributed systems.

Michael Reiter is Director of Secure Systems Research in Bell Laboratories, Lucent Technologies. He received the B.S. degree in mathematical sciences from the University of North Carolina in 1989, and the M.S. and Ph.D. degrees in computer science from Cornell University in 1991 and 1993, respectively. He joined AT&T Bell Labs in 1993 and became a founding member of AT&T Labs Research when NCR and Lucent Technologies (including Bell Labs) were split away from AT&T in 1996. He returned to Bell Labs in 1998 to take his current position. His research interests include all areas of computer and communications security, and distributed computing.