On the Availability of Non-strict Quorum Systems*

Amitanand Aiyer[1], Lorenzo Alvisi[1], and Rida A. Bazzi[2]

[1] Department of Computer Sciences,
The University of Texas at Austin
{anand, lorenzo}@cs.utexas.edu

[2] Computer Science and Engineering Department,
Arizona State University,
bazzi@asu.edu

Abstract. Allowing read operations to return stale data with low probability has been proposed as a means to increase availability in quorums systems. Existing solutions that allow stale reads cannot tolerate an adversarial scheduler that can maliciously delay messages between servers and clients in the system and for such a scheduler existing solutions cannot enforce a bound on the staleness of data read. This paper considers the possibility of increasing system availability while at the same time tolerating a malicious scheduler and guaranteeing an upper bound on the staleness of data. We characterize the conditions under which this increase is possible and show that it depends on the ratio of the write frequency to the servers' failure frequency. For environments with a relatively large failure frequency compared to write frequency, we propose K-quorums that can provide higher availability than the strict quorum systems and also guarantee bounded staleness. We also propose a definition of k-atomicity and present a protocol to implement a k-atomic register using k-quorums.

1 Introduction

Quorum systems have been extensively studied in the literature. A traditional, or strict, quorum system is simply a collection of sets called quorums such that any two quorums have a non-empty intersection. Quorum systems have been used for mutual exclusion, coordination, and data replication in distributed systems.

In a particular protocol using quorum systems, quorums are accessed either to write a new value to a quorum or to read the values stored in a quorum. Important quality measures of quorum systems are *fault tolerance*, *availability*, *load*, and *quorum size*. In general, these quality measures are conflicting in strict quorum systems [13]. Systems with high availability tend to have large quorum sizes and high load. If the failure probability of individual nodes is less than 0.5, the system with the highest availability is the majority system in which a quorum

* This work was supported in part by NSF CyberTrust award 0430510, an Alfred P. Sloan Fellowhip and a grant from the Texas Advanced Technology Program.

consists of a majority of the servers. Unfortunately, this system suffers from high load and large quorum size, which means that a large subset of servers need to be up in order for the system to be usable. However, in environments such as peer-to-peer networks, where the availability of individual nodes is not very high, the availability of the majority system is not high. If the failure probability of a node is more than 0.5, the best system in terms of availability is the singleton, but that system has a very high load [14].

In order to develop quorum systems with small quorum sizes and with availability higher than that of the majority system, probabilistic quorum systems have been proposed [11]. A probabilistic quorum system is a collection of sets together with an access strategy which specifies the probability that a quorum is chosen to be used in an access. In a probabilistic quorum system, two quorums chosen according to the access strategy have a non-empty intersection with probability $1 - \epsilon$, where ϵ is a system parameter. Using probabilistic quorum systems, a read access is guaranteed to get the value of the most up-to-date non-overlapping write access with probability $1 - \epsilon$.

More recently, Yu [20] proposed Signed Quorum systems which aim at overcoming problems with the definition of probabilistic quorum systems. Yu observed that probabilistic systems cannot be realized in a system in which the scheduler is an active adversary that delays responses from servers to prevent clients from following the probabilistic access strategy. He proposed a signed quorum system (SQS) in which the high probability of intersection depends on the assumptions that the probability that two clients observe conflicting (mismatch) states (up or down) of servers is low and that simultaneous mismatches of different servers are independent—the independent mismatch assumption. These assumptions are backed by trace results from a number of experiments [3,21]; further, Yu argues that in practice probabilistic quorum systems would require making explicit assumptions about the scheduler similar to the ones SQS makes. In SQS, a quorum consists of positive elements (servers that respond and are up) and negative elements (servers that do not respond and are assumed to be down). By allowing servers that are down to be part of a quorum, Yu's system can be used even if a small number of servers are up.

Both probabilistic systems and SQS make implicit or explicit assumptions about the scheduler [20]. Both systems would not perform as claimed if these assumptions do not hold. Both systems are unusable in the presence of an adversarial scheduler that controls the delay to various nodes in the system. In fact, in the presence of an adversarial scheduler, the returned values can be arbitrarily old. Also, due to their probabilistic nature, both probabilistic systems and signed quorum systems do not provide strict guarantees on the freshness of values returned by a quorum. With positive, albeit small, probability, a quorum might return a value that is old (even in the absence of an adversarial scheduler) and there is no way for a client to tell how old the value is.

This paper investigates the following question. "Is it possible to design a quorum system that: provides strict guarantees on the staleness of values returned, tolerates an adversarial scheduler, and, in the absence of an adversarial scheduler provides higher availability than the majority system?" It turns out that the answer to this question does not only depend on the nodes' failure probability, but it also depends on the rate at which write operations are executed, the mean time between failure ($mtbf$), and the mean time to recover ($mttr$) of individual nodes. We prove a lower bound on the possible increase in availability as a function of the staleness of values and the ratio of the frequency of writes to the frequency of failures ($1/mtbf$). We show that for some values of the ratio, the possible increase of availability is negligible. For the cases where the increase in availability is not negligible, we propose K-quorums, which can have higher availability (for the same system size) than that of the majority system when the system is well behaved (no adversarial scheduler) and that have bounded staleness in the presence of an adversarial scheduler. Our study of bounded-staleness also led us to revisit the properties of signed quorum systems. For signed quorum systems, we found that these systems are not guaranteed to behave as predicted in [20] if the times between writes is large compared to the times between failures.

In summary, we achieve the following in this paper.

- We prove a lower bound on the availability of systems that can tolerate an adversarial scheduler and provide guarantees on the staleness of returned values.
- We introduce K-quorum systems which, for some combination of system parameters, have lower load and higher availability than traditional quorums systems.
- We show how to use K-quorum systems for providing K-atomic implementations of a shared register.

The rest of the paper is organized as follows. Section 2 presents the probabilistic approaches to increasing the availability of quorum systems and discusses their limitations. Section 3 introduces K-consistency semantics, a formalization of relaxed access semantics with bounded staleness. Section 4 presents the system model and the definition of traditional quorum systems. Section 5 proves lower bounds on the availability of quorum system with worst-case guarantees on staleness. Section 6 introduces K-quorum systems and shows how they can be used to implement K-atomic registers. Section 7 gives an overview of related work.

2 Probabilistic Approaches

In strict quorum systems any read and write quorum sets have a non empty intersection. This allows for easy construction of registers with safe-semantics, where any read – that is not concurrent with any write – is guaranteed to return the value from the latest write.

Probabilistic approaches, such as PQS [11] and SQS [20], provide a high availability and low load at the cost of weakening consistency semantics. In these systems, the read and write quorums only intersect probabilistically. Hence, these systems can only provide safe-semantics probabilistically. Probabilistic guarantees can cause these systems to return arbitrarily old values. In synchronous systems, the probability that a read violates safe-semantics can be made arbitrarily small by using a large quorum size with an appropriate access strategy. However there can be no bound on this probability in an asynchronous system where an adversarial scheduler can affect the choice of quorums.

2.1 Probabilistic Quorum System

A probabilistic quorum system consists of read and write quorums similar to strict quorums, along with an access strategy for choosing quorums [11]. Any two quorums that are chosen according to the specified access strategy will intersect with a high probability.

In a simple construction of such a quorum system with n nodes, quorums are chosen to be sets of cardinality $l\sqrt{n}$ where l is a system parameter [11]. For large values of n, these systems provide a higher availability than the majority quorum system because they require only $l\sqrt{n}$ nodes to be accessible, as opposed to requiring $\frac{n+1}{2}$ in the majority quorum system. In a synchronous setting, using a uniform random access strategy guarantees that any two quorum sets of size $l\sqrt{n}$ intersect with probability at least $1 - e^{l^2}$. However, in an asynchronous system where the scheduler may be adversarial the probability of intersection can be much smaller – in fact it can be zero.

Examples. Consider a probabilistic quorum system over nodes $\{1, 2, \ldots, 100\}$, where any set of 30 nodes form a quorum. In the presence of an non-adversarial scheduler, if quorums are chosen uniformly at random, then the probability that two quorums do not intersect is less than 1.88×10^{-6}. However in an asynchronous system in which the scheduler arbitrarily delays read messages to $\{1, 2, \ldots, 50\}$ and also delay write messages to $\{51, 52, \ldots, 100\}$, read and write quorums will never intersect causing reads to always return arbitrarily old values.

2.2 Signed Quorum Systems

Signed quorum systems (SQS) [20], like PQS, provide a probabilistic guarantee of intersection. SQS utilizes the notion of a failure detector to form an estimate of nodes that may be inaccessible. Quorums in SQS consist of both positive and negative elements. Positive elements denoting the servers that have been contacted, and negative elements denoting servers that have been suspected to fail. Two quorums are said to intersect if and only if they intersect in a positive element. Like PQS, SQS also increases the availability by allowing non-intersecting quorums. However, SQS requires that no two non-intersecting quorums be accessible in the same configuration. In fact, SQS requires that the configurations in which two non-intersecting quorums are accessible differ in at least 2α node-states.

In a system with perfect failure-detectors, if the configuration of the nodes does not change (or less than 2α nodes change state), then SQS can always guarantee safe-semantics and behave like a strict quorum system. The probability of non-intersection is equal to the probability that more than 2α nodes used by a read and write access have different states and this probability is lower for larger values of α. In a asynchronous system, it is not possible to distinguish a failed node from a node whose messages are all delayed by the adversary (assuming that the only means to determine the state of a node is through message exchanges). In such a setting, an adversarial scheduler can present the reader and writer with totally different configurations so that the quorums used never intersect. A reader may then read an arbitrarily old value.

3 K-Consistency Semantics

The semantics of shared objects that are implemented with quorum systems can be classified as *safe*, *regular* or *atomic* [9]. For applications that can tolerate some staleness these notions of consistency are too strong. We propose the notion of K*-safe*, K*-regular* and K*-atomic* semantics, similar to those defined in [9], for formalizing consistency semantics in applications that can tolerate limited staleness.

1. K**-safe:** In a system that provides K*-safe* semantics, a read that does not overlap with a write is guaranteed to return the result of one of the latest K completed writes. The result of a read, that overlaps with a write is unspecified.
2. K**-regular:** A system that provides K*-regular* semantics, guarantees that any read, that does not overlap with a write, is guaranteed to return the result of one of the latest K completed writes. A read that overlaps with a write, returns either the result of one of the latest K completed writes, or the eventual result of one of the overlapping writes.
3. K**-atomic:** In a system with K*-atomic* semantics, there exists an order of the operations that is consistent with real time order and such that the values returned by a read operation is equal to one of the values written by the last K preceding writes in the order (assuming there are K initial writes with the same initial value).

4 Model and Definitions

4.1 Model

The system consists of n nodes, $\mathcal{P} = \{1, 2, \ldots, n\}$, each of which may be inaccessible with a probability p_f. Nodes are assumed to crash and recover independently, with a mean-time-to-failure of $mttf$ and a mean-time-to-recover of $mttr$. The mean-time-between-failures, is $mtbf = mttr + mttf$ and $p_f = \frac{mttr}{mtbf}$. If a node is up, then it is assumed to follow the specified protocol; i.e. we assume there are no malicious faults. Each node is assumed to have access to stable storage, such that the values written to the servers are persistent across crashes. The system is assumed to be asynchronous, with no bound on the relative speeds of the nodes. The links are modeled as *fair links*, i.e. if a message is sent infinitely often, it will eventually be delivered at the receiver.

In this paper, we assume that only servers fail. We assume that the duration of operations are small enough so that we can neglect the client failures during the operations. Our results, for the single writer multiple reader scenario, can however be easily extended to tolerate benign client-failures by incorporating a logging protocol at the client end.

4.2 Quorum Systems

Definition 1. *A* quorum system *over the set of nodes $\mathcal{P}$ is a tuple $(\mathcal{R}, \mathcal{W})$; where $\mathcal{R} \subset 2^{\mathcal{P}}$ is the set of read quorums and $\mathcal{W} \subset 2^{\mathcal{P}}$ is the set of write quorums.*

During a read (write) operation, the reader (writer) contacts a read quorum (write quorum) to perform a read (write) operation. The access strategy specifies which nodes need to be contacted to access a quorum set.

Definition 2. *An* access strategy *for a client specifies an algorithm for choosing a quorum set to access, possibly based on the previous local history at the client.*

For strict quorum systems, the access strategy allows the system to contact any of the quorums, as long as every write quorum intersects with a read quorum. In probabilistic quorum systems, the access strategy is probabilistic and the quorum is chosen at random, ignoring the local history at the client. In Section 6 we present protocols which provide stronger non-probabilistic consistency guarantees, using an access strategy that is dependent on the client's local history.

Definition 3. *A* configuration *C of a system specifies the state of each node in the system (either as accessible or inaccessible).*

For systems with independent failures, the probability of the system being in a configuration C, with K accessible nodes, is $P(C) = \binom{n}{k}(1 - p_f)^k {p_f}^{n-k}$.

An operation of a client is *successful* in a given configuration if the quorum set that should be accessed as specified by the access strategy (or one of the quorum sets, if the strategy specifies more than one valid set) consists of elements that are all available.

In what follows we define the availability for systems in which the scheduler is not adversarial. The availability of the system is only defined during periods in which the system is well behaved, i.e. periods where any message sent from a non-crashed node to another non-crashed node is guaranteed to be delivered within a fixed (may be unknown) time bound. If the network is asynchronous then an adversarial scheduler can delay all messages arbitrarily to stall any system from making progress, hence making the system unavailable and any definition of availability meaningless.

Definition 4. *The availability of the system for reads, $a_r(C, t)$, in a given configuration C over a time interval of duration t, is defined as the ratio of successful reads to the total number of reads in an interval of time of length t when the system is in the specified configuration C.*

Definition 5. *The availability of the system for reads, $a_r(C)$, in a given configuration C is the probability that a read operation is successful when the system is in configuration C.* [1]

Let t be the random variable denoting the duration of a configuration C and whose probability distribution is determined by the failure behavior of the nodes. Let $t_1, t_2, \ldots$ denote the various realizations (time durations of configuration C) of t in an execution. If $succ_i$ denotes the number of successful reads, and tot_i denotes the number of attempted reads in the i^{th} realization, then

$$a_r(C) = \lim_{m \to \infty} \frac{\sum_{i=1}^{m} succ_i}{\sum_{i=1}^{m} tot_i}$$

Definition 6. *The availability of the system for reads, a_r is defined as the expected availability, $E[a_r(C)]$, over all the possible configurations.*

We define the availability of the system for writes, $a_w(C,t)$, $a_w(C)$, and a_w on lines similar to the availability definitions for reads.

Definition 7. *The availability of a Quorum System, is defined as the fraction of successful operations when the network is synchronous.*

Our definitions are a generalization of the definitions used previously in [11,14,20]. In the traditional quorum systems and probabilistic quorum systems, where the access strategy is independent of the local history, the availability $a_r(C,t)$ and $a_w(C,t)$ will be independent of t. However, this may not be the case if the access strategy is dependent on the local history.

5 Bounds on Increase in Availability

Consider a quorum system Q which provides a bounded staleness of K. For any configuration C, let $w_Q(C)$ be the write availability and $r_Q(C)$ be the read availability of the quorum system in the configuration C.

Let γ_r and γ_w be the rates of read and write operations in the system, and let $\tau(C)$ be the expected duration of a configuration C (we assume that the rates of read and writes are constants, but the results still apply by replacing the rates with expected rates).

Lemma 1. *If C is a configuration with l nodes that are up and $n-l$ nodes that are down, then the expected duration of the configuration, $\tau(C)$, is $\geq \frac{1}{\frac{l}{mttf}+\frac{n-l}{mttr}}$*

Proof. Consider a small duration of time dt. The probability of particular node, that is currently crashed, recovering during an interval of length dt is $\frac{dt}{mttr}$. Similarly, the probability of a node crashing is $\frac{dt}{mttf}$.

[1] This definition does not depend on the distribution of read operations if that distribution is independent from that of system configurations.

In configuration C, there are l nodes that are up and $n-l$ nodes that are down. Therefore the probability that the system, currently in configuration C, changes to some other configuration during an interval of time of length dt is $\leq \left(\frac{l}{mttf} + \frac{n-l}{mttr}\right)dt$. Hence, the expected duration for which a configuration lasts, $\tau(C)$, is $\geq \frac{1}{\frac{l}{mttf}+\frac{n-l}{mttr}}$.

For systems where nodes are available with a probability > 0.5, $mttf > mttr$ and

$$\tau(C) \geq \frac{1}{\frac{l}{mttf} + \frac{n-l}{mttr}} \geq \frac{1}{\frac{l}{mttr} + \frac{n-l}{mttr}} = \frac{mttr}{n} = \tau_{min}$$

The expected duration of any configuration is at least $\tau_{min} = \frac{mttr}{n}$. Let $AC(K)$ denote the set of all configurations C such that K writes can be executed successfully in C and a read can be executed successfully after K writes are executed in C.

Lemma 2. *For any configuration $C_i \in AC(K)$, there exist two sets $W(C_i)$ and $R(C_i)$ that are available during the configuration C_i and, $W(C_i) \cap R(C_i) \neq \emptyset$.*

Proof. If $C_i \in AC$, it follows that there can be K successful writes in the configuration C_i and a successful read after the k^{th} write. Let R be the read quorum that was used for the successful read after the k^{th} write. Let $W_1, W_2, \ldots, W_K$ be the set of servers contacted during the K successful writes. Since the system provides bounded staleness, it follows that

$$R \cap \bigcup_{j=1}^{j=k} W_j \neq \emptyset$$

Choose $W(C_i) = \bigcup_{j=1}^{j=k} W_j$ and $R(C_i) = R$. Since each W_j and R is available in C_i it follows that $W(C_i)$, and $R(C_i)$ are available in C_i.

Lemma 3. *For any two configurations, $C_i, C_j \in AC(K)$, $W(C_i) \cap R(C_j) \neq \emptyset$.*

Proof. Consider the configuration in which all nodes are accessible. Since the scheduler can be adversarial, it can arbitrarily delay messages from the writer to all nodes in $\mathcal{P} \setminus W(C_i)$, hence forcing the next K writes to be written to nodes in $W(C_i)$. Later, it can delay the messages from the reader to the nodes in $\mathcal{P} \setminus R(C_j)$, so that the reader is forced to choose $R(C_j)$ as the read quorum. Since the read is guaranteed to return one of the K latest written values, it follows that $W(C_i) \cap R(C_j) \neq \emptyset$

Lemma 4. *Let C be a configuration. If $C \notin AC(K)$, then $w_Q(C) + r_Q(C) \leq 1 + \epsilon$, where $\epsilon = \frac{k}{\gamma_w \tau_{min}}$.*

Proof. If $C \notin AC(K)$, there are two cases: either there are no more than K writes that can occur in C, or there is no read that can occur after the K^{th} write in C.

– If there are no more than K writes that can occur in C, then

$$\begin{aligned} w_Q(C) &= \lim_{m\to\infty} \frac{\sum_{i=1}^m succ_i}{\sum_{i=1}^m tot_i} \le \lim_{m\to\infty} \frac{mK}{\sum_{i=1}^m tot_i} \\ &= \lim_{m\to\infty} \frac{mK}{m\gamma_w \tau(C)} = \frac{K}{\gamma_w \tau(C)} \le \frac{K}{\gamma_w \tau_{min}} \\ w_Q(C) + r_Q(C) &\le \frac{K}{\gamma_w \tau_{min}} + 1 = 1 + \epsilon \end{aligned}$$

– Let t be the time in the configuration by which the K^{th} write succeeded. If there are no successful reads after the K^{th} write, then all reads up to t can succeed but all later reads fail. Also there are at most K writes succeeding up to t, and writes after time t may succeed.

$$\begin{aligned} r_Q(C) &\le \frac{t}{\tau(C)} \\ w_Q(C) &\le \frac{K + \gamma_w\big(\tau(C) - t\big)}{\gamma_w \tau(C)} \\ w_Q(C) + r_Q(C) &\le \big(1 + \frac{K}{\gamma_w \tau(C)}\big) \le \big(1 + \frac{K}{\gamma_w \tau_{min}}\big) = 1 + \epsilon \end{aligned}$$

Consider the quorum system $Q'(\mathcal{W}', \mathcal{R}')$, where $\mathcal{W}' = \{W(C_i) | C_i \in AC(K)\}$, and $\mathcal{R}' = \{R(C_i) | C_i \in AC(K)\}$. Q' is a strict quorum system, that is available in all configurations $C \in AC(K)$.

Theorem 1. *The read and write availability of the strict quorum system, Q' is $\ge r_Q + w_Q - 1 - \frac{2\epsilon}{1-\epsilon}$*

Proof. The strict quorum system Q' is available, for both reads and writes, during any configuration $C \in AC(K)$. Let $P(C)$ denote the probability that the system is in configuration C, and let $P_{AC(K)} = \sum_{C\in AC(K)} P(C)$. The read availability of Q', $r_{Q'} \ge \sum_{C\in AC(K)} P(C) = P_{AC(K)}$. Similarly, $w_{Q'} \ge P_{AC(K)}$. For the bounded-staleness quorum system, Q,

$$\begin{aligned} w_Q = \sum P(C) w_Q(C) &= \sum_{C\in AC(K)} P(C) w_Q(C) + \sum_{C\notin AC} P(C) w_Q(C) \\ &\le \sum_{C\in AC(K)} P(C) + \sum_{C\notin AC(K)} P(C) w_Q(C) \end{aligned}$$

$$\begin{aligned} r_Q &\le \sum_{C\in AC} P(C) + \sum_{C\notin AC} P(C) r_Q(C) \\ w_Q + r_Q &\le 2 \sum_{C\in AC} P(C) + \sum_{C\notin AC} P(C)\big(r_Q(C) + w_Q(C)\big) \\ &\le 2 \sum_{C\in AC} P(C) + \sum_{C\notin AC} P(C)\big(1 + \epsilon\big) \\ &\le 2P_{AC} + (1 - P_{AC})(1 + \epsilon) \end{aligned}$$

Therefore,

$$P_{AC} \geq \frac{w_Q + r_Q - 1 - \epsilon}{1 - \epsilon} > w_Q + r_Q - 1 - \frac{2\epsilon}{1 - \epsilon}$$

It follows that $w_{Q'}, r_{Q'} > w_Q + r_Q - 1 - \frac{2\epsilon}{1-\epsilon}$

Theorem 1 shows that the increase in availability due to relaxing the consistency guarantees to K-safe is dependent on the rate of write operations γ_w, and the expected duration τ_{min} of a configuration. τ_{min} increases *linearly* with the mean-time-between-failure for the nodes. Hence, for a large *mtbf*, the value of $\epsilon = \frac{K}{\gamma_w \tau_{min}}$ will be small. In such cases, if highly available K-safe quorum systems can be built, then a highly available strict quorum systems can also be built i.e. there is not much advantage gained by relaxing the consistency semantics to K-safe. However, for systems with a small *mtbf*, it may be possible to increase the availability and at the same time provide bounded staleness. We show a protocol to achieve this in section 6.

6 *K*-Quorums Protocols

We present K-quorum construction, for a single-writer-multiple-reader environment, which guarantees bounded staleness even in the presence of an adversarial scheduler. In Section 6.2, we prove that the proposed protocol achieves K-atomic semantics.

6.1 Construction and Protocols

A K-quorum system consists of a strict quorum system, $(\mathcal{R}, \mathcal{W})$, and a staleness parameter K that is the bound on the staleness allowed.

Read operations in K-quorums are similar to reads in strict quorum systems. At a high level, the reader contacts a quorum of servers $R \in \mathcal{R}$ and chooses the latest value. The writes are different. In K-quorums, a value is written to a subset of $\mathcal{P}$, such that the servers contacted during K consecutive writes form a write-quorum $W \in \mathcal{W}$. We henceforth call the set of servers contacted during a particular write a *partial-write-quorum.*

The single-writer-multiple-reader protocol for K-quorums, is shown in figure 1. For simplicity, the protocol presented assumes reliable channels. The protocol can be made to work with fair channels by using standard techniques, for building a reliable channels over fair channels, as described in [12].

Write operation. To perform a write operation the writer chooses a partial-write-quorum, and writes the value along with other meta-data to the partial-write-quorum. To ensure bounded staleness we require that any K partial-write-quorums, used for successive writes, collectively contain a write quorum. Formally, let W_i be the partial-write-quorum used in the i^{th} write. We require that

$$\forall i : \exists W \in \mathcal{W} \textit{ such that } W \subseteq \bigcup_{j=i-K+1}^{i} W_j$$

```
// Writer Protocol
static $k$ := 0; static $ts$ := 0;
void Write( v )
begin
    $ts := ts + 1$; $k := k + 1$;
    find an available partial-write-quorum $W_k$ such that $\exists W \in \mathcal{W} : W \subseteq \bigcup_{i=k-K+1}^{i=k} W_i$
    send $(v, ts, PW)$ to servers in $W_k$, where $PW = \bigcup_{i=k-K+1}^{i=k-1} W_i$
    wait for acknowledgments from servers in $W_k$
end

// Reader Protocol
int Read
begin
    find an available read quorum $R$
    send read requests to servers in $R$
    wait for replies from all servers
    calculate $(v, ts, PW)$ := value with the largest time stamp
    write back the value $(v, ts, PW)$ to a partial-write-quorum, $W_r$
        such that $\exists W \in \mathcal{W} : W \subseteq PW \cup W_r$
    wait for acknowledgments from servers in $W_r$
    return(v, ts, PW)
end
```

Fig. 1. K-quorum protocols

The protocol for write is shown in Figure 1. During the i^{th} write, the writer writes the value v, the timestamp – ts, and the set PW of servers accessed in the previous $(K-1)$ writes to each of the servers in W_i.

Read operation. Reads in K-quorums are similar to reads in strict quorum systems. The reader collects replies from a quorum of servers and chooses the reply (v, ts_{hst}, PW) with the largest time stamp. The reader then writes back the tuple (v, ts_{hst}, PW) to a set of servers W' such that $\exists W \in \mathcal{W} : W \subseteq PW \cup W'$. The protocol for a read is shown in Figure 1. Since a read quorum always intersects with one of the previous K partial-write-quorums, a read is guaranteed to return one of the K latest written values irrespective of the behavior of the scheduler.

6.2 K-Atomic Semantics

To prove that the protocols achieve $K-atomic$ semantics, we show the existence of an ordering that is consistent with real time order such that, each read returns the value written by one of the previous K writes. We define

Definition 8. *The* written-time *is the (global) time at which a value that is being written reaches (and is processed by) every server in partial-write-quorum.*

We will order the reads and writes such that :

- Writes are ordered according to their *written-time.*
- A read which returns a value (v, t, PW), which was written with timestamp t, can be scheduled any time between
  1. The written-time, τ_t of the value returned, (v, t, PW).
  2. and, before the written-time of the next K^{th} write, $(v', t+K, PW')$. i.e. before τ_{t+K}.

It is easy to see that, such an ordering satisfies the requirements of K-atomic semantics. We need to show that such a ordering can be done in a manner consistent with local history.

The scheduling of writes is trivial, because written-time of a write occurs between the time a write has begun and before the write ends.

We now show, by contradiction, that reads can also be scheduled. Assume, if possible, that the read interval does not overlap with the interval $[\tau_t, \tau_{t+K})$. There are two cases:

1. Read finishes before τ_t:
   This scenario is not possible, because a read completes only after performing a write-back on the value. Therefore a read can end only after the written-time of the value it returns.
2. Read begins after τ_{t+K}.
   Consider the union of the partial write quorums for K previous writes – $W = W_{t+1} \cup W_{t+2} \cup \ldots \cup W_{t+K}$. From the definition of a partial-write-quorum and the fact that any read quorum intersects with a (complete) write quorum, it follows that the reader would have received a value from at least one server in W. Since the reader chooses the highest time-stamp received, a read that starts after τ_{t+K} cannot return a value written before τ_K, which is a contradiction. $\Rightarrow\Leftarrow$

Theorem 2. *Protocols described in Figure 1 provide K-atomic semantics* $\square$

6.3 Availability

In environments with a relatively small *mtbf*, K-quorum systems can be used to achieve a higher availability than strict quorum systems. Consider a K-quorum system with staleness parameter K and where the read and write quorum sizes are rn and wn respectively. Let $r = rn/n$ and $w = wn/n$. Since the read and write (not partial write) quorums need to intersect, we require $r + w \geq \frac{n+1}{n}$.

For a read to be available, we need rn nodes out of n nodes. This can be made strictly smaller than the availability of majority if $r \leq 0.5$ and the goal is to simultaneously make the write availability be better than that of majority. For a system, where *mtbf* is *relatively* small, the write availability is the probability of being able to access $\frac{wn}{K}$ nodes out of $\left(n - \frac{K-1}{K}wn\right)$ nodes (the availability of a given write is independent of that of previous writes). For a given n and p_f, with appropriate choice of r,w and K, even the write availability of the system can be increased. For example, consider a case where $n = 100$ and $p_f = 0.5$. The majority quorum system will be available with a probability 0.46. With K-quorums, using a read quorum of size 29, a write quorum of size 72 and a staleness bound of $K = 6$ can provide much better availability for both reads and writes. Reads are available with probability 0.99999 (29 out of 100 available), while writes are available with probability 0.997 (12 out of 40 available). Also, if the system is well behaved the probability that a read will get the value of the most recent write is 0.99.

For a given system size n, probability of failure p_f and staleness parameter K, choosing the optimal values for r and w presents similar trade-offs as in strict quorum system. For having a large read availability it is desirable to have a small value for r. Similarly, for having a large write availability, it is desirable to have a small value for w. However as we require that $r + w \geq \frac{n+1}{n}$, choosing a small value for r or w will require the other to be large, resulting in decreased write or read availability respectively. For optimal overall availability, this trade-off needs to be resolved based on the relative frequencies of reads and writes in the system, such that the overall availability is maximized.

Effect of *mtbf* on Signed Quorum Systems. Realizing that the performance of K-quorum systems is highly dependent on the mean time between failure, we investigated their effect on the performance of systems such as signed quorum systems. We found that the performance of signed quorum systems deteriorates in environments in which the mean-time-between-failures is smaller than the mean time between a write and a read. One explanation for this is that if the system has a small mean time to failure, then nodes go up and down very quickly, so it is more likely that the reader and the writer see two different configurations of the system, thereby causing mismatches. For RON traces, where there are no node failures, the probability of mismatch due to network faults alone was found to be 0.05. However, in systems where *mtbf* is small to the mean time between a write and a read, the probability of mismatch can be much higher even when ignoring network faults.

Lemma 5. *In a system with a small mtbf, if nodes are inaccessible with a probability p_f, then probability of mismatch ignoring any network faults is $2p_f(1-p_f)$.*

Proof. (sktech) For systems with a small *mtbf*, the expected duration of a configuration is small. So, the configuration of the system can change widely between the time a value is written to the system and the time when the value is read. For this setting the probability that a server if down during the read operation is independent of the probability of the server is down during the write operation. The probability that a node is accessible during a read, but not accessible during a write is $(1 - p_f)p_f$. Similarly the probability that a node is accessible during a write, but not accessible during a read is $p_f(1 - p_f)$. Therefore the probability of mismatch is at least $2p_f(1 - p_f)$, which can be as high as 0.5 depending on the value of p_f.

7 Related Work

There is a very large body of work on quorum systems and access semantics. In this section we concentrate on those works that are most closely related to the results of this paper.

Quorum systems are used for various distributed applications including replication, mutual exclusion, and consensus. The performance measures for quorum systems like load, availability and probe complexity, are mutually opposing – improving one tends to worsen the other. With large scale internet usage, researchers have been mainly focusing on improving the availability of the system [1,2,4,6,18,19]. Naor and Wool prove some basic bounds between the load and the availability of traditional quorum systems [13].

Lamport presents consistency semantics that have been widely used in the literature [9] . Traditional quorum systems, where any two quorum sets have a non-empty intersection, provide at least safe-semantics. However, these systems are not very highly available. Fox and Brewer [5] show bounds on availability in the presence of strong consistency guarantees.

Relaxing the consistency guarantees can allow systems to achieve better availability or performance. For database applications, a number of researchers ([8,16] for example) discuss weakened consistency semantics for increased concurrency. Epsilon consistency [16] attempts to increase availability by allowing query accesses to see some temporary inconsistencies in the data, however these inconsistencies are bounded and the system converges to a global serializability. Krishnamurthy et al present bounded ignorance [8], for increasing the concurrency in database applications, where the application may be unaware of at most N transactions. [7,15,17] implement file systems that provide various relaxed semantics. TACT is a toolkit that allows for dynamic changes in the consistency level of the system and can be used to specify various kinds of weakened semantics.

Probabilistic approaches to quorum systems [11,20] achieve a much higher availability than strict quorum systems by weakening the consistency. Lee and Welch [10] propose probabilistic relaxed semantics for use with probabilistic system. These systems provide a probabilistic bound on the violation of safe-semantics. However, as discussed in section 2, these systems are vulnerable to an adversarial scheduler and provide no bounds on the staleness of data.

References

1. D. S. Amr El Abbadi and F. Cristian. An efficient and fault-tolerant protocol for replicated data management. In *Proc. PODS*, 1985.
2. D. S. Amr El Abbadi and F. Cristian. Maintaining availability in partioned replicated databases. In *Proc. PODS*, 1986.
3. D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris. Resilient overlay networks. In *Proc. 18th SOSP*, pages 131–145, 2001.
4. D. L. Eager and K. C. Sevcik. Achieving robustness in distributed database systems. *ACM Trans. Database Syst.*, 8(3):354–381, 1983.
5. A. Fox and E. A. Brewer. Harvest, yield and scalable tolerant systems. In *Wkshp on Hot Topics in Op Sys*, pages 174–178, 1999.

6. D. K. Gifford. Weighted voting for replicated data. In *Proc. 7th SOSP*, pages 150–162, New York, NY, USA, 1979. ACM Press.
7. R. G. Guy, J. S. Heidemann, W. Mak, T. W. Page, Jr., G. J. Popek, and D. Rothmeir. Implementation of the Ficus Replicated File System. In *Proceedings of the Summer 1990 USENIX Conference*, pages 63–71, June 1990.
8. N. Krishnakumar and A. J. Bernstein. Bounded ignorance in replicated systems. In *Proc. PODS*, pages 63–74, New York, NY, USA, 1991. ACM Press.
9. L. Lamport. On interprocess communication. part i: Basic formalism. *Distributed Computing*, 1(2):77–101, 1986.
10. H. Lee and J. L. Welch. Randomized registers and iterative algorithms. *Distributed Computing*, 17(3):209–221, 2005.
11. D. Malkhi, M. K. Reiter, A. Wool, and R. N. Wright. Probabilistic quorum systems. *Inf. Comput.*, 170(2):184–206, 2001.
12. J.-P. Martin, L. Alvisi, and M. Dahlin. Small Byzantine quorum systems. In *DSN*, pages 374–383, June 2002.
13. M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, 1998.
14. D. Peleg and A. Wool. The availability of quorum systems. *Inf. Comput.*, 123(2):210–223, 1995.
15. K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Proc. 16th SOSP*, 1997.
16. C. Pu and A. Leff. Replica control in distributed systems: An asynchronous approach. In *SIGMOD Conference*, pages 377–386, 1991.
17. M. Satyanarayanan, J. J. Kistler, P. Kumar, M. E. Okasaki, E. H. Siegel, and D. C. Steere. Coda: A highly available file system for a distributed workstation environment. *IEEE Transactions on Computers*, 39(4):447–459, 1990.
18. D. Skeen and D. D. Wright. Increasing availability in partitioned database systems. In *Proc. PODS*, pages 290–299, New York, NY, USA, 1984. ACM Press.
19. H. G.-M. Susan Davidson and D. Skeen. Consistency in partioned network. *Computing Survey*, 17(3), 1985.
20. H. Yu. Signed quorum systems. In *Proc. 23rd PODC*, pages 246–255. ACM Press, 2004.
21. H. Yu and A. Vahdat. The costs and limits of availability for replicated services. In *Proc. 18th SOSP*, pages 29–42, 2001.