ANALYZING AVAILABILITY OF REPLICATED DATABASE SYSTEMS

Bharat Bhargava
Abdelsalam Helal
Karl Friesen

CSD-TR-91-013
February 1991
Analyzing Availability of Replicated Database Systems*

Bharat Bhargava
Abdelsalam Helal
Karl Friesen

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
(317) 494-6013

Abstract

Qualitative studies have shown that replication control methods vary in the availability and performance of distributed database processing. Quantitative evaluation of these methods, however, requires a general availability model and experimental performance data. In this paper, we define and study algorithmic and operational availabilities of distributed database systems that employ replication as a technique to achieve fault-tolerance. We show that both availability definitions are complementary and therefore, should be studied simultaneously. We present a customer-stationary availability measure that includes four basic sets of parameters in its underlying model. These are transaction, data, configuration and failure parameters, in addition to parameters pertinent to the replication control method itself. We study algorithmic availability of the read-one-write-all and the quorum consensus replication control methods through a series of experiments in transaction processing of the RAID distributed database system. Operational availability of the same methods is studied using the SETH distributed database prototype and an event-driven simulator.

*This research is supported in part by a grant from AIRMICS, a David Ross Fellowship, a Purdue University Fellowship, and an NSF grant IRI-8821398
1 Introduction

Replicated copies of databases provide higher availability in distributed database systems. By maintaining multiple copies of data objects, some copies of the database remain available even though the system has suffered site or communication link failures. However, the addition of multiple copies increases the amount of communication and computing resources required to execute a transaction. Moreover, replication requires a consistency criterion that ensures that transactions have the same effect in a replicated database as they would in a single-copy database, even in presence of failures. One-copy serializability [BG83] has been used to define the correctness of a replicated database. A replication control method is used to enforce one-copy serializability. Many replication control methods have been proposed [BG84, AD76, MW82, Gil79, AT86, Her86, Hed88]. They vary in the way they affect the performance and the degree of availability. Research on performance and availability evaluation of these methods is still in its infancy.

This paper studies availability in distributed database systems that use data replication. By availability we mean the probability that a transaction succeeds in completing all its operations even in the presence of failures. In particular, we study the availability of two replication control methods: the read-one-write-all (ROWA) and the quorum consensus (QC) methods. Details of these two methods are explained in Sections 2.2.1 and 2.2.2. The paper introduces two different, yet complementary, definitions of availability. Algorithmic availability defines a measure of merit through which replication control methods can be theoretically evaluated, regardless of their implementation. Operational availability, on the other hand, is measured by the range and conditions under which a “no-failure” distributed database system is operable. Replication control methods that are algorithmically highly available may be expensive to implement and hence become operationally less available. In other words, a replication control method can be theoretically highly available, but in practice, this availability may be achievable only under restricted conditions. For this reason, we study both types of availability simultaneously.

To study the algorithmic availability of the ROWA and the QC methods, we derive a customer-stationary availability measure that includes four basic sets of parameters in its underlying model. These are transaction parameters (such as transaction size and the ratio of operations that are reads to those that are updates), data parameters (such as the degree of replication and data directories), configuration and failure parameters (such as network topology and site and link reliability), and parameters pertinent to the replication control method itself (such as the size of the read and write quorums). We study algorithmic availability through a practical experiment that examines both the performance and availability of the RAID distributed database system [BFH+90]. We use the availability measure that we derived in Section 2 along with experimental performance data to examine the effect of the degree of replication on both availability and performance. Our study aimed at finding the lowest degree(s) of replication at which both performance and availability requirements
were realized. The details of this study are presented in Section 3.

To study operational availability, we use the SETH distributed database prototype [HSB89] to measure the maximum transaction load for which SETH can be operable under the quorum consensus replication control method. We extend the study by using simulation to investigate the scalability of transaction processing power to the increase in the maximum message queue length of the underlying communication network.

The paper is organized as follow. Section 2 elaborates on availability and gives the details of the availability model. A practical study on algorithmic availability in the context of the RAID system is presented in section 3. Operational availability is studied in section 4, in the context of the SETH prototype and through simulation. Our conclusions and remarks are summarized in section 5.

2 Availability

In this section we present a probabilistic availability model for replicated database systems. We start by reviewing the classical definition of availability and showing how it differs from reliability. We then specialize the discussion on the availability of replication control methods that are used by replicated database systems. We survey related studies on analyzing availability before we give the details of our availability model.

2.1 The Classical Definition of Availability

The reliability, $R(t)$, of a system is defined as the probability that the system’s constituent components are functioning properly in the interval $[0 \ldots t]$. In other words, $R(t)$ is the probability that the system’s lifetime exceeds $t$, given that the system was functioning properly at time 0. Thus, $R(t)$ can be determined from the system failure density function, $f(x)$, and is

$$R(t) = 1 - F(t) = 1 - \int_0^t f(x)\,dx$$

$R(t)$ can also be measured experimentally by observing the system states over a long period of time $[0 \ldots t]$, and recording the periods of time, $u_i$, where the system was up. Reliability is then stated as

$$R(t) = \frac{\sum_i u_i}{t}$$

The interval $[0 \ldots t]$ over which the system is observed is chosen equal to the utilization interval of the system, usually called the mission time [Che88].

Failure repair and/or component redundancy can be used to extend system reliability beyond expected component lifetimes. The enhanced reliability that results from repairs and redundancy, is defined as the probability that either:
\[ A(t) = \frac{\mu}{\lambda + \mu} + \frac{\lambda}{\lambda + \mu} e^{-(\lambda + \mu)t} \]

\[ R(t) = e^{-\lambda t} \]

mean time to failure = \( \frac{1}{\mu} = 0.1 \)
mean time to repair = \( \frac{1}{\lambda} = 1.0 \)

Figure 1: Reliability vs. Availability

- the system has been functioning properly in the interval \([0 \ldots t]\), or
- the last failed component has been repaired or redundantly replaced at time \( x \), \( 0 < x < t \), and the repaired(redundant) component has been functioning properly since then (in the interval \([x \ldots t]\)).

This enhanced reliability is called availability, \( A(t) \). In [Tri82], \( A(t) \) is called the instantaneous availability and \( \lim_{t \to \infty} A(t) \) is defined as the limiting availability. Obviously, \( A(t) \geq R(t) \), and \( \lim_{t \to \infty} A(t) \geq \lim_{t \to \infty} R(t) \). The limiting availability is shown to depend only on the mean time to failure and mean time to repair, and not on the nature of the distributions of failure times and repair times [Tri82]. Figure 1 depicts the difference between the reliability and instantaneous availability as a function of time, for exponential failure and repair distributions.

In distributed database systems, replication of data is employed to further enhance system availability during periods of failure. When part of the system fails or becomes inaccessible, data access that would normally be made to that part can be directed to the replicas on the available parts of the database. In order to allow updates, data redundancy requires a
special protocol that guarantees all redundant copies to converge to an identical up-to-date value. Unfortunately, the availability of the update operation can be severely affected when many replicas fail or become inaccessible. In this case, the protocol (called the replication control protocol) aborts the transaction issuing the update. Updates can, however, succeed despite of site failures or network partition. This happens in the case where the failed or inaccessible part of the system does not contain redundant copies. Therefore, even though the accessibility of the system improves with increasing the degree of data redundancy, the availability of the system for updates decreases. Therefore, the amount of data redundancy in the system is a crucial parameter that must be chosen carefully.

2.2 Database Systems' Availability

Transaction response time, system throughput, and volume of message traffic are well-defined and accepted metrics that are used to measure the performance of distributed database systems. However, the availability of data for transaction processing remains an inadequately defined metric. This inadequacy stems from the fact that the classical definition of availability can not capture the essential aspects of distributed database systems primarily designed for transaction processing. In this section, we define two classes of database system’s availability: algorithmic and operational availability.

Algorithmic availability defines a measure of merit through which replication control methods can be theoretically evaluated, regardless of any specific implementation of these methods or their system counterparts. In section 2.4, we present a general model of algorithmic availability.

Operational availability defines the range and conditions under which a replicated database system is not operable, even in the absence of failures. These include the maximum allowed transaction processing rate (or equivalently, the maximum allowed degree of multiprogramming), maximum allowed log size (used by log-based replication methods, and usually stored on limited-capacity local disks), and the probability of deadlock occurrence in systems that do not handle deadlocks. For example, a client-server implementation of a distributed database system that uses the UNIX user datagram protocol (UDP) sockets to communicate lacks flow control of exchanged messages. Such implementation can experience message loss when the total message traffic exceeds the maximum message processing power. This could be the case with communication-intensive replication methods such as the quorum-based methods [Gif79, Her86, Kum90, Hed88]. Section 4 examines operational availability and presents a study on the effect of the finiteness of UDP queue length on the maximum possible transaction load.

In the following subsections (sections 2.2.1 and 2.2.2), we qualitatively discuss the availability of two basic replication control methods that are studied in this paper. These are the read-one-write-all and the quorum consensus methods.
2.2.1 The Read-One-Write-All Method

The read-one-write-all (ROWA) is the simplest replication control method. In this method, a read operation is directed to a single copy (usually, the local copy), while a write operation is performed on all copies. The main features of the ROWA method are discussed as follows:

- **Availability**: The availability of the write operations is very poor. Under full replication, a single site failure is sufficient to block the write operations. On the other hand, the availability of the read operations is very high since any available copy can be used. Write availability can be increased by employing partial replication and/or by buffering updates to unavailable sites in the communication subsystems [R+80]. Another approach is to write only the available copies and bring stale copies up-to-date when their sites recover from failures. This approach does not work in presence of network partitions. The available copies [BG84] and the read-one-write-all-available [Bha87] methods follow this approach.

- **Message Traffic**: The read operations incur lower message traffic compared to the write operations. However, the technique of deferred writes [BH87], whereby all the write operations of a transaction are deferred till commit time, substantially reduces the message traffic associated with the write operations.

- **Computation Cost**: The computation cost attributed to message processing is much smaller for the read operations than for the write operations. Deferred writes can reduce the cost due to message traffic. In the deferred writes case, the computation cost attributed to local processing of transactions is much smaller for the writes than for the reads, especially for read-dominant transactions. This is because every read request has to be processed separately whereas all the writes are processed as a single request.

2.2.2 The Quorum Consensus Method

The quorum consensus (QC), (or weighted voting) [Gif79] is a general mechanism for managing replicated data objects. Under quorum consensus each data object is assigned a read threshold and a write threshold. Also, each copy of the data object is assigned a weight, and a version number. In order to read/write a data object, a read/write quorum must be available. Any set of available copies with total weight greater than or equal to the read/write threshold constitutes a read/write quorum. Weights are chosen so that any write quorum of a data object intersects any other read and write quorum of the same object. This is called the quorum intersection rule. The rule guarantees that any pair of conflicting operations will have quorums that intersect and therefore will always be synchronized by the concurrency control at the sites on which the intersected copies reside. The most up-to-date copy of the read quorum is used for the read. This is the copy with the highest version number. When
writing the value for an object. all copies of the write quorum are assigned a version number greater than their current maximum version number. Quorum consensus can be adapted to implement read-one-write-all (ROWA), read-all-write-one (RAWO), or read-same-as-write (QC-RSW) policies [BFHR90]. The main features of the quorum consensus method are discussed as follows:

- **Availability**: Quorum consensus can achieve higher availability than the ROWA method. This is because it is sufficient that a quorum of copies be available to perform an operation. Unlike the ROWA method, quorum consensus availability can be tuned to favor the read operation over the write operation. Furthermore, this tuning can be done on per object basis.

- **Message Traffic**: Quorum consensus requires access to multiple remote copies in order to process a read operation. Unfortunately, the quorum sizes increases linearly with the number of copies for an object. This leads to heavy message traffic that can become a bottleneck in the communication subsystem. The hierarchical quorum consensus [Kum90] method can improve scalability by reducing the message traffic overhead. This is done by organizing the copies in a multi-level hierarchy, and requiring that a quorum for an object at a certain level in the hierarchy be assembled by gathering a vote of sub-objects at the previous level in the hierarchy.

- **Computation Cost**: The computation cost attributed to message processing is proportional to the quorum size for each operation on each data object. Deferred writes reduce the computation cost of processing messages. To reduce the cost of the read operations, read requests of the same transaction can be piggypacked in the communication subsystem. For example, consider a transaction with 5 read operations three of which have site 4 as a member of their quorums. A single message that contains three read requests can be sent to site 4, instead of three separate messages. Unfortunately, per-site piggybacking requires a periori knowledge of the read set of transactions.

### 2.3 Related Studies on Availability

One approach to analyzing availability uses the up/down system modeling (also called: the $k$-out-of-$N$ system modeling) [BD84, GMK87]. In this approach, system components can be in only one of two states: up or down. The up/down system modeling is a discrete stochastic markovian modeling where the database access operation is a markov process over a space of $2^n$ up/down states, for an n-site distributed database system. The up/down modeling approach is, however, at a rather abstract level of transaction management description. As such, this approach can not adequately take into account parameters such as partial replication, fault-tolerant replication methods, transaction classes and types, and operation mix, etcetera. For example, with partial replication, up/down system modeling does not
differentiate between the two cases where a failed site contains or does not contain a copy of the data object. The approach is also restrictive to the types of failures, where the communication link components of the system are assumed to be reliable.

A less restrictive approach to evaluate availability was proposed in [MRS81]. In this approach, partial replication, transaction types, and network topology were included. The approach depends on a methodology that requires a static analysis of both transactions and the network topology. The analysis creates a search graph that contains a number of nodes from the source to the sink, which correspond to a transaction's requirements for performing its read and write operations. This graph is compiled into a discrete markov chain that is used to compute the steady state probability that a transaction successfully completes. The methodology, however, requires a prohibitive amount of static data analysis that grows exponentially with the product of the number of sites and number of replicas in the system. Moreover, the methodology pertains only to the read-one-write-all replication method, and therefore can not be used to evaluate other methods without modifications. Similar to the up/down modeling approach, the search graph approach is restrictive to the types of failures where communication links are assumed to be reliable.

Continuous time stochastic markov analysis was used to study and compare the performance of fault-tolerant replication control methods. The difference between this approach and the up/down and the search graph approaches is in the modeling emphasis given by these studies to the replication control method itself. The reliability of a regeneration-based available copies algorithm was presented in [LCS89] where object reliability was defined to be the probability that the object remains continuously accessible over a given time period. The trade-off between storage space and reliability was examined. The same approach was taken in [JM87] to analyze the dynamic voting algorithm.

Customer-stationary probabilistic analysis is an approach that can be used to analyze availability. A simple comparative availability study was presented to compare hierarchical quorum consensus [Kum90] against majority consensus [Tho79]. The analysis assumed reliable communication links. In another study, an upper-bound on the availability when a network partitions into a majority and a minority partitions was presented in [COK86]. The upper-bound was, however, based on an estimate of availability that was defined to be the ratio between the number of transactions successfully completed to the number of transactions presented to the system. A similar experimental estimate has been used in SETH [HSB89], where availability after a single point failure was measured as the ratio of the sum of throughput at each partition to the throughput of the same system before the failure point.

In the next section we present a customer-stationary probabilistic availability model that does not assume full reliability of the communication links and that can be used to model most replication control method. Our model captures transaction parameters like transaction sizes and read/write mix, database parameters like the degree of replication and data directories, configuration parameters like the network topology and failure types and
probabilities, and parameters pertinent to the replication control method itself like the size of the read and write quorums.

2.4 An Availability Model

Definition: We define algorithmic availability, \( \alpha \), as the probability that an arbitrary transaction \( t \) successfully starts and successfully finishes execution. A transaction succeeds in starting if all the objects it needs are accessible. The transaction succeeds in finishing if it either commits, or aborts only due to concurrency conflicts.

Consider a distributed database system whose sites and communication links can be modeled by a graph \( \langle \Sigma, \Lambda \rangle \). Let \( \Sigma(G) \) be a function that returns the set of nodes of a graph \( G \). Let \( \Theta = \{\theta_1, \theta_2, \ldots, \theta_d\} \) be a set of \( d \) objects that are replicated over \( |\Sigma| \) sites according to a fully replicated data directory \( D \). \( D(\theta_i) \) is then the directory entry for object \( \theta_i \) that contains the list of sites where the object is replicated.

Transactions are classified into classes according to their sizes. Consider a transaction \( t \) that belongs to class \( c \in C \). Assuming no particular replication control method, the transaction availability, \( \alpha \) is given by:

\[
\alpha = \sum_{c \in C} Pr\{t \text{ succeeds} | t \in \text{class } c\} Pr\{t \in \text{class } c\}
\]  

(1)

Let \( n(c) \) be the number of data objects actually accessed by transactions of class \( c \). That is, \( n(c) \) is the size of class-\( c \) transactions. Also, let \( \{\theta_{k_1}, \theta_{k_2}, \ldots, \theta_{k_{n(c)}}\} \) be the set of accessed objects that is determined according to certain access distribution (uniform, hot-spot, etc.).

Given that \( A_{\theta_i} \) is the availability of object \( \theta_i \), the probability that transaction \( t \) in class \( c \) succeeds is given by:

\[
Pr\{t \text{ succeeds} | t \in \text{class } c\} = \left( \prod_{i=1}^{n(c)} A_{\theta_i} \right) Pr\{t \text{ finishes}\}
\]  

(2)

Let \( Pr\{\text{operation} = \rho\} \) be the probability that the operation on object \( \theta_i \) is \( \rho \). \( A_{\theta_i} \) is then given by:

\[
A_{\theta_i} = \sum_{\rho \in \Omega} Pr\{\theta_i \text{ is available} | \text{operation} = \rho\} Pr\{\text{operation} = \rho\}
\]  

(3)

where \( \Omega \) is the domain of operations, which is \{read, write\} in a simple transaction model.

Let \( Pr\{\text{site} = \sigma\} \) be the probability that transaction \( t \) was issued at site \( \sigma \). Then:
Let \( \Pi_\sigma = \{ \pi_1, \pi_2, \ldots \} \) be the set of all possible partitions such that \( \sigma \in \pi_i, \forall \pi_i \in \Pi_\sigma \). Then, \( \Pi_\sigma \) induces the set of subgraphs \( \mathcal{G} \subseteq < \Sigma, \Lambda > \) which contains \( \sigma \) and which arises due to site and communication failures. The probabilities of site and communication failures define the probability that site \( \sigma \in \) partition \( \pi \). Therefore:

\[
\Pr\{\theta_i \text{ is available for } \rho \mid \text{operation} = \rho \} = \sum_{\sigma \in \Sigma} \Pr\{\theta_i \text{ is available for } \rho \mid \text{site} = \sigma\} \Pr\{\text{site} = \sigma\}
\]

\[
(4)
\]

Then:

\[
\Pr\{\theta_i \text{ is available for } \rho \mid \text{site} = \sigma\} = \sum_{\pi \in \Pi_\sigma} \Pr\{\theta_i \text{ is available for } \rho \text{ at site } \sigma \mid \text{Partition}(\sigma) = \pi\} \Pr\{\text{Partition}(\sigma) = \pi\} = \sum_{\pi \in \Pi_\sigma} \beta_{\theta_i, \rho, \sigma, \pi} \cdot \gamma_{\sigma, \pi}
\]

\[
(5)
\]

where \( \beta_{\theta_i, \rho, \sigma, \pi} \) is the probability that object \( \theta_i \) is available for operation \( \rho \) issued at site \( \sigma \), given that site \( \sigma \) is in partition \( \pi \), and where \( \gamma_{\sigma, \pi} \) is the probability that site \( \sigma \) is in partition \( \pi \). \( \gamma_{\sigma, \pi} \) is easy to compute. However, generating all possible partitions \( \pi \) for general graphs is a hard problem.

Let \( \mathcal{RC} \) be a particular replication control method, then \( \mathcal{RC}(\theta_i, \rho, \sigma, \pi) \) denotes the set of all possible quorums that can satisfy operation \( \rho \) on object \( \theta_i \) from site \( \sigma \) on partition \( \pi \). In practice, even though many quorums could be available, only a few of them are selected due to performance reasons. For example, the smallest-size quorums are usually preferred by quorum selection heuristics. To compute the probability \( \beta_{\theta_i, \rho, \sigma, \pi} \), however, we are interested in enumerating all possible quorums rather than just the practical ones. \( \beta_{\theta_i, \rho, \sigma, \pi} \) is then:

\[
\beta_{\theta_i, \rho, \sigma, \pi} = \frac{|\mathcal{RC}(\theta_i, \rho, \sigma, \pi)|}{|\mathcal{RC}(\theta_i, \rho, \sigma, \pi_0)|}
\]

\[
(6)
\]

Where \( \pi_0 \) is the initial partition (the connected system). At this point in the derivation, knowledge of the specific replication control method is needed in order to compute \( \beta_{\theta_i, \rho, \sigma, \pi} \). We demonstrate the evaluation of \( \beta \) for two replication control methods: the read-one-write-all (ROAW) and the quorum consensus (QC) methods.

For \( \mathcal{RC} = \text{ROAW} \), and for a simple read/write transaction model:
\[ R_{\text{ROWA}}(\theta_i, \rho, \sigma, \pi) = \begin{cases} \{s_i\} : s_i \in \Sigma(\pi) \cap D(\theta_i) \}, & \rho = \text{read}, \\ \{D(\theta_i)\}, & \rho = \text{write} \land \Sigma(\pi) \cap D(\theta_i) = D(\theta_i), \\ \emptyset, & \text{otherwise}. \] (7)

For \( R_C = QC \) (with equal weights \( w \)), and for a simple read/write transaction model, let \( R_{\text{thresh}}(\theta_i) \) and \( W_{\text{thresh}}(\theta_i) \) be the read and write thresholds of object \( \theta_i \). Then:

\[ |QC(\theta_i, \rho, \sigma, \pi)| = \begin{cases} \left( \frac{\left| \Sigma(\pi) \cap D(\theta_i) \right|}{R_{\text{thresh}}(\theta_i)} \right), & \rho = \text{read}, \\ \left( \frac{\left| \Sigma(\pi) \cap D(\theta_i) \right|}{W_{\text{thresh}}(\theta_i)} \right), & \rho = \text{write}, \\ 0, & \text{otherwise}. \] (8)

The software that evaluates the model reads the model parameters along with the data directory and computes availability based on a replication method specifier. The model parameters include the update percent, the transaction length, and the workstation reliability, among other parameters. Quorum thresholds and weights are read in the case of quorum consensus. In section 3, we use the model evaluation software with \( Pr\{t \text{ finishes}\} = 1.0 \), for the ROWA and the QC methods. The model is used to obtain availability data for various degrees of replication and for different workstation reliabilities. The obtained availability data will be used with other performance measurements of the RAID system to study the effect of the degree of replication on the availability-performance tradeoff of replicated database systems.

3 Algorithmic Availability: An Experimental Study

In this section, we study the algorithmic availability and the response time of the read-one-write-all and the quorum consensus replication control methods. The study aims at finding the lowest degree of replication for which acceptable availability and performance are realized.
The study was performed on the second version of the RAID distributed database system [BR89, BFH+90]. An action driver simulator was used to generate transactions that follow a variation of the DebitCredit benchmark [Ano85]. The RAID experimental infrastructure [BFHR90] was used to specify and automatically execute experiments, and to gather, analyze, and plot performance data. The availability model presented in section 2 is used in this study. In what follows, we give an overview of the second version of the RAID system and its experimental infrastructure and action driver simulator. Next, we present the details of the experimental study and its findings and conclusions.

3.1 RAID Overview

RAID is a distributed database system that is being developed on Sun workstations under the Unix operating system. Each database site in a RAID system consists of six servers, each of which encapsulates a subset of the functionality of the system. The six servers are the User Interface (UI), the Action Driver (AD), the Access Manager (AM), the Concurrency Controller (CC), the Atomicity Controller (AC), and the Replication Controller (RC). The user interface allows users to interact with the system via a query language. The action driver translates the user-level instructions into a sequence of low-level read and write actions. The access manager is responsible for the storage of information on a physical device. The
Table 1: RAID Execution Time (in ms)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of Sites</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Select 1 tuple</td>
<td>197.5</td>
</tr>
<tr>
<td>Insert 1 tuple</td>
<td>150.2</td>
</tr>
<tr>
<td>Update 1 tuple</td>
<td>214.8</td>
</tr>
<tr>
<td>Update 2 tuples</td>
<td>270.6</td>
</tr>
</tbody>
</table>

concurrency controller checks that read and write actions of different transactions do not conflict. The atomicity controller is responsible for ensuring that transactions are committed or aborted uniformly across all sites. The replication controller manages multiple copies of data items to provide system reliability and mutual consistency of the data. Figure 2 illustrates the paths of communication in RAID.

The RC provides on-line replication control for the system. It supports partial replication as well as full replication. The RC may be configured to execute one of several replication control policies ranging from the simple Read-One-Write-All (ROWA) method, to the general Quorum Consensus (QC) algorithm. Quorum assignments are determined by a special relation that contains the quorum parameters. Many of the standard replication methods (such as ROWA) may be expressed using this method. In addition, the RC may dynamically alter quorum assignments during transaction processing and thus adapt to changes in the operating environment.

Table 1 lists the cost of performing basic transaction operations on one, two, three, and four sites RAID running on Sparcstation1's. The basic timestamp method was used by the CC; two-phase commit was used by the AC; and read-one-write-all was used by the RC. Each measurement represents the average response time of the corresponding operation.

3.2 RAID Experimental Infrastructure

Our experiments were conducted on the RAID distributed database system running on five Sun 3/50s and 4 Sun Sparcstation-1s connected by a single 10MB/s Ethernet segment. Each of the machines had a local disk.

In addition to RAID, we wrote support programs to specify, create, and maintain databases. These allowed us to specify relation attributes and replication information such as the number of copies that are created for each object and how these copies are distributed throughout the system.

The transaction stream for our experiment is generated by an action driver simulator as an extended version of the DebitCredit benchmark [Ano85]. The extensions allowed us to specify parameters such as: the number of generated transactions, the average transaction length, update percent, hot-spot size, and hot-spot access percent.
Throughout the experiments presented in this section, the two-phase commit protocol and the two-phase locking concurrency control were used.

3.3 Availability and performance for various degrees of replication

The effect of the degree of replication on the availability and response time of replicated database systems is examined in this experiment. Data for the availability is obtained via the software that evaluates the availability model of section 2.4. Data for the response time is actual measurements when transactions are executed in RAID.

3.3.1 Statement of the Problem

A high degree of replication incurs performance penalties. This is due in part to the number of messages required to access larger number of copies. Even so, a high degree of replication does not always increase data availability. These series of experiments investigate the effect of the degree of replication on the availability-performance tradeoff in face of changing operating conditions.

3.3.2 Procedure

The experiments were run early morning to minimize the effects of network traffic from external sources. All of the machines were rebooted before the experiments were started to provide a consistent, uniform operating environment. After the reboot, an experiment specification file was read and a new RAID instance was started. Upon completion of the provided transaction stream, each server wrote its performance statistics to a log for the experiment. A new specification file and new RAID instance were used for the next experiment. After an experiment was finished, it was "checked-out" through a group of shell scripts that:

1. digest all log files of the experiment into a global log that contains high level statistics like response time, throughput, commit time, etc.,

2. compute the confidence interval of dependent variables under examination, and

3. plot the necessary graphs of the experiment.

We have conducted the experiment for two replication methods: the read-one-write-all (ROWA) and a special case of the quorum consensus method where reads and writes have the same quorum parameters (QC-RSW, or, Reads Same as Writes). For each method, we measured the average response time and analytically computed the database availability against various degrees of replication. We then identified degrees of replication at which availability was maximum and those at which response time was minimum. Where possible,
we also identified the lowest degree of replication for which availability was within 10% of its maximum and for which response time was within 15% of its minimum. We call this the \textit{practical degree of replication}.

3.3.3 Data

The experiments that measured the response time were run on a 9-site database. The degree of replication was varied from 1 to 9. We could not experiment with higher degrees of replication since we have only 9 machines in the Raid laboratory. Transaction size was fixed at 6 operations with update fractions of 0%, 20%, 50%, and 100%. With less than 6 operations per transaction, the effect of the update percent was not clearly observable. The database hot spot was 20% of the database, and the maximum degree of multiprogramming was fixed at 3. Higher degrees of multiprogramming were avoided since they increased the variance in response time due to increased transaction restart rate. In each measurement, we executed 250 transactions (not including restarts) and computed their average response time. Each data point of the experiment was obtained by averaging at least 10 independent measurements. Confidence interval analysis was used to test the acceptability of the average. Accordingly, some experiments were repeated for improvements.

Availability was evaluated according to the model presented in section 2. Model parameters that are also RAID parameters are given the same value by the model evaluation software. The workstation reliability (the probability that the workstation is up at any given time) was varied from 0.95 to 0.80.

Figures 3, 4, and 5 show the response time and availability of the ROWA method for a 0, 20, and 50 update percents and for workstation reliability of 0.90. Figures 8, and 6 show the response time of the QC-RSW method for a 0 and 50 update percents and for workstation reliability of 0.90. Figure 9 shows the response time and availability of the same method for 50 update percent and for workstation reliability of 0.80. Figure 7 shows the response time and availability for 50 update percent and for workstation reliability of 0.95.

3.3.4 Discussion

- \textit{ROWA Availability:} For read-only transactions (0% updates) and a workstation reliability of 0.90 (see Figure 3), the response time was almost constant (2 seconds) for all degrees of replication. Availability was lowest (0.28) when the degree of replication was 1 and highest (0.53) when the degree of replication was 3 or greater. The practical degree of replication was 2 copies. Higher degrees of replication did not improve availability or impair the response time. Therefore, in a 9-site system and with the mentioned access pattern and reliability assumptions, 2-copy replication is as good as 9-copy replication.

When the fraction of updates was increased to 20 percent (see Figure 4), the practical
Figure 3: ROWA: 0% Updates, 0.90 Reliability

Figure 4: ROWA: 20% Updates, 0.90 Reliability
Figure 5: ROWA: 50% Updates, 0.90 Reliability

Figure 6: QC-RSW: 50% Updates, 0.90 Reliability
Figure 7: QC-RSW: 50% Updates, 0.95 Reliability

Figure 8: QC-RSW: 0% Updates, 0.90 Reliability
degree of replication was again 2 copies. The practical degree of replication was a unique maximum. Therefore, higher replication should be avoided since it impairs availability in this case.

When 50% of the actions were updates (see Figure 5), the practical degree of replication (also a unique availability maximum) was 1 copy. In this case, replication was not helpful.

- **ROWA Response Time:** In our experiments, the response time of the ROWA method is not sensitive to the degree of replication. This is because the response time mainly consists of the time spent in processing the read operations (which does not depend on the number of replicas), regardless of the update percent. To explain this observation, we consider both cases of low and high update percents. For low update percents, the response time is an accumulation of the response time of the sequential execution of the read operations on local copies. This time is obviously independent of the number of replicas. On the other hand, writes are deferred into a single update operation that takes place at all replicas in parallel. Due to deferring writes and to the parallelism, no matter how high the update percent is, there will always be only a single update operation that involves all replicas and that has a chance to execute in parallel.

- **QC-RSW Availability:** For transactions that were 50% updates and workstations that had a 0.90 reliability factor (see Figure 6), the response time was almost constant (4.2
seconds) when the degree of replication was less than 6. For higher degrees of replication, response time increased linearly to a maximum of 10 seconds when the degree of replication was 9. The availability of QC-RSW had an interesting behavior. The availability oscillated around a slowly increasing average that reached a plateau when the degree of replication was seven or greater. Even degrees of replication consistently had less availability than the preceding odd degrees of replication. Therefore, 2 copies are less available than 1 copy. Three copies are more available than 1 and 2 copies. Four copies are less available than three copies, and so forth. This zigzag effect can be explained as follows. Consider a 3-site system completely connected with each site having a reliability factor 0.9. For a degree of replication of 1, the probability that a designated site is up is 0.9 (note that the probability that any one site is up is higher). For a degree of replication of 2 (in which QC-RSW requires both copies), the probability that 2 designated sites are up is 0.81 (less than 0.9). For a degree of replication of 3 (in which QC-RSW requires any 2 copies), the probability that any 2 sites are up is the sum of the probabilities that the configurations 011, 101, 110, and 111 occur. This amounts to $3 \times (0.9^2) + 0.9^3$ or 0.972 (greater than 0.9 and 0.81). As can be inferred from the model, such probabilities directly affect availability, and are responsible for the zigzag behavior. Figure 6 shows that availability was lowest (0.15) for a degree of replication 2, and highest (0.53) at a degree of replication of 9. The practical degree of replication was 5 copies. Higher replication had a detrimental effect on response time and should be avoided.

For higher workstation reliability (0.95) and under same transaction mix (see Figure 7), the practical degree of replication decreased to 3 copies. Availability was lowest (0.41) at a degree of replication of 2, and highest (0.725) when there were 9 copies.

Availability remained unchanged when the transaction mix was varied. This is easy to explain since QC-RSW requires the same number of sites for both the read and write operations. For read-only transaction (0% updates), and a workstation reliability factor of 0.90 (see Figure 8), the availability was identical to the case depicted in Figure 6 where half of the operation were updates.

When updates comprised 50% percent of the actions and the workstation reliability was decreased to 0.80 (see Figure 9), the availability did not exceed 0.25 (at 9 copies) and almost diminished (at 2 copies). Moreover, there was no practical degree of replication in this case. That is, either response time or availability must be compromised. For example, to maintain an availability level greater than 0.2, 7 copies or more would be needed. This resulted in poor performance (7 - 10 seconds).

- **QC-RSW Response Time:** The response time of the QC-RSW method shown in Figures 8, and 6 increases with the degree of replication. This is because higher the higher the degree of replication, the bigger the size of the read majority quorums, and therefore,
the higher the message traffic.

4 Operational Availability: An Experimental Study

Operational availability defines the range and conditions under which a replicated database system is not operable, even in the absence of failures. In this section, we study the operational availability of the quorum consensus replication method (QC-RSW) whose algorithmic availability was examined in section 3. We use the SETH distributed database prototype [HSB89] and a distributed database simulator for this study. We examine the overhead incurred by the quorum consensus method and measure the maximum attainable transaction processing power. This is the transaction load beyond which the quorum consensus overhead hinders transaction processing to the extent that makes the system unavailable, thus defeating the purpose of replication. The study consists of two parts. In the first part, we give an overview on the SETH prototype, briefly describe its experimental setup, and give the details of the SETH experiment. The second part extends the SETH experiment by a simulation study. The purpose of the extended study is to investigate the effect of varying hardware and communication parameters on the allowed transaction load, while keeping the system available for transaction processing.

4.1 SETH Overview

SETH is a quorum-based replicated database prototype. Its design emphasizes facilitating experimentation with various patterns of site and communication link failures. SETH virtual sites are UNIX processes that process concurrent transactions through three layered protocols: a replication control protocol (quorum consensus), an atomic commitment protocol (two-phase commit), and a concurrency control protocol (simple two-phase locking). Database objects are replicated as copies in the sites' virtual memory with a varying degree of replication. SETH assumes no particular communication network type. In other words, it is designed to be network independent. This required the implementation of a network interface called the prototype manager. The interface helped in separating the design of most of the prototype from the underlying communication network model.

Figure 10 shows an instance of SETH which consists of a number of SETH sites, a prototype manager, and a workload generator. Figure 10. The prototype manager (or simply, the manager) maintains information about the system's configuration. SETH sites communicate only through the manager, and hence need not worry about view information, or view synchronization. Also, sites need not know much about the underlying network type and its characteristics. In essence, the manager acts as a message forwarder.

The manager, maintains view information through an adjacency matrix and an up-down vector. In addition, it computes and maintains the closure of the adjacency matrix which
contains the least number of hops needed to go from one site to another. When the manager receives a message from a site, it consults the closure matrix to determine the number of hops the message needs to travel. It then relays the message to the destination site after delaying it by a time proportional to the number of hops. The manager may receive a message that has to be forwarded to a site which is unreachable. In such a case, it either drops the message or sends a NACK reply.

The workload generator generates transactions to SETH sites and failure patterns to the prototype manager. Transaction and failure parameters are specified in a file that is read by the workload generator.

SETH sites and the prototype manager are implemented as UNIX processes that communicate through UDP sockets. When a SETH site is initiated, it selects a UDP port on the local host machine and writes it into a well-known file. Other sites can then read this file to determine the port to which messages should be sent in order to communicate. SETH communication subsystem is cannibalized from the communication library of RAID [BFH+90]. In addition, it includes a traffic monitor that gathers message statistics and detects network congestion.

4.2 The SETH Experiment

4.2.1 Statement of the Problem

The quorum consensus replication control method can provide high availability at the cost of increased communication overhead. For high transaction load, the overhead can increase
to the extent that may disallow further transaction processing. In this experiment, we measure the quorum consensus overhead against a spectrum of transaction loads. We identify the maximum attainable transaction load beyond which SETH is said to be operationally unavailable.

4.2.2 Procedure

To set up the experiment, three sets of parameters were specified. These are: the network configuration, quorum parameters, and transaction and failure workload. Specifying a network configuration involves specifying the number of sites and the sites' adjacency matrix. Quorum parameters (weights and thresholds) are specified for each site using an interactive quorum-parameters design module that validates the parameters with respect to the quorum intersection rules. The transaction workload is parameterized by the transaction arrival rate, the maximum transaction size, and the transaction's read/write ratio (or mix). The transaction arrival process is assumed to be Poisson. Transaction size is uniform over [1…maxmum transaction size]. The transaction read/write ratio is specified by the probability of reads. The failure workload is prescribed as a list of failure or repair events. Each event consists of an adjacency matrix and the relative time at which the new view goes into effect.

4.2.3 Data

We ran this experiment on a 6-processor Sequent Symmetry machine. A database of 200 objects was fully replicated over 8-site fully connected configuration. Copies were assigned equal weights. We conducted the experiment for transaction average arrival rates varying from 0.5 to 4.0 transactions per second. The transaction size was chosen to be uniform over [1-5]. The read/write ratio was fixed to 0.5. For each arrival rate, we measured the average message traffic rate. Each measured point was actually an average of 6 identical experiments. Each experiment included 150 transactions. Figure 11 shows the average message traffic rate against the average transaction arrival rate.

4.2.4 Discussion

Figure 11 shows how the message rate sharply increases as the arrival rate increases from 0.5 to 2.0 transactions per second. However, the message rate starts to saturate at 2.5 transactions per second. For example, doubling the arrival rate from 0.5 to 1.0 doubles the traffic from 35 to 70 Messages per second, while increasing it from 3.5 to 4.0(by 14%) does not increase the message traffic but slightly decreases it, instead. To explain this behavior, we recognize that the maximum rate at which messages can be processed in a machine like the Sequent Symmetry is 200 messages per second. For arrival rates greater than 2.5, the message traffic approaches the maximum message processing rate. This results in congesting the
sockets' queues. In this case, sockets start dropping messages. This explains the saturation in the message traffic curve. Furthermore, some of the dropped messages could be request messages for which transactions will be waiting on their replies. Since replies will never arrive, those transactions will be blocked and will not contribute to further message traffic. This explains the small decline of the message rate for transaction arrival rates of 3.5 and 4.0. The conclusion that we can draw in this experiment is that one requirement for SETH to be operationally available, is not to allow transaction loads higher than 2.0 transactions per second.

4.3 Extending the SETH Experiment by Simulation

The SETH experiment has shown that the overhead incurred by the quorum consensus method along with the finiteness of the UDP queues set a limit on the maximum possible transaction load that SETH can afford. An inspection of the maximum UDP queue length on the Sequent's UNIX kernel revealed a constant of 34 entries. The same constant was only 7 for the Sun3 kernel. In order to study the effect of the UDP queue length on the maximum possible transaction load, we needed to vary the kernel constant and repeat the SETH experiment for each such constant. This process would have required recompiling the Sequent kernel every time we change that constant. For administrative reasons, we could not have the authority to pursue this direction. Therefore, we decided to resort to open-system simulation to investigate our quest. The simulation approach was attractive since we also wanted to study the sensitivity of the impact of the finiteness of the UDP queue length to the underlying network latency. In the following subsections, we describe the simulator and
its parameters and assumptions, and we present a simulation experiment and its results.

4.3.1 The Simulator

The simulator models replicated database systems that consist of a set of sites connected by a local area network. For simplicity, each site is assumed to consist of a single local disk besides the CPU. Data are assumed to be fully replicated. A transaction consists of a set of Read and Write operations. For an operation to be carried on, a quorum is required. Quorum consensus is coded into the simulator to form quorums. Figure 12 depicts the simulator queuing model of one site in the system. Each site has 4 queues: a cpu queue, an io queue, a network queue and a queue that is used as a waiting list. The waiting list is used to model the blocking of a transaction that is waiting for its subtransactions in other sites to finish and report back to it. Any transaction that enters the waiting list schedules a timeout event for itself. The timeout event is canceled upon arrival of all subtransactions. The network queue was chosen to have a finite capacity so that it models the UDP sockets on UNIX.

The simulator does not model time-sharing preemptive scheduling. Instead, a job consists of a sequence of cpu and io bursts. A transaction workload takes the form (cpu, io, cpu, wait, cpu, io, cpu, wait, ...). That is, for its first operation, a transaction does local cpu processing, then reads its local values, then uses cpu to build up subtransaction messages. It then puts the subtransactions in the network queue and blocks itself in the waiting list for the responses regarding the operation or the timeout event. When all responses arrive, the transaction is removed from the waiting list and the next operation is considered by going through another cpu/io/cpu/wait cycle. An operation is determined to be read or write probabilistically. A subtransaction consists of (cpu, io, cpu) after which the subtransaction answer(in the form of a message) will be put on the network queue and the subtransaction dies at that point. Concurrency control is macro-simulated by a probability of conflict on
Table 2: Simulator Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$</td>
<td>number of sites in the system</td>
<td>3</td>
</tr>
<tr>
<td>mst-cpu</td>
<td>mean (exponential) service time of the CPU</td>
<td>0.01</td>
</tr>
<tr>
<td>mst-io</td>
<td>mean (exponential) service time of the Disk</td>
<td>0.10</td>
</tr>
<tr>
<td>mst-net</td>
<td>mean (exponential) service time of the LAN network</td>
<td>0.25, 0.50</td>
</tr>
<tr>
<td>mat</td>
<td>mean (Poisson) arrival rate of transactions all over the sites</td>
<td>0.2 – 8.0</td>
</tr>
<tr>
<td>rwr</td>
<td>the Read-to-Write ratio, or the Prob. of an operation being Read</td>
<td>0.5</td>
</tr>
<tr>
<td>mTsiz</td>
<td>maximum transaction size</td>
<td>5</td>
</tr>
<tr>
<td>maxtr</td>
<td>maximum number of transactions in the system</td>
<td>100</td>
</tr>
<tr>
<td>TIMEOUT</td>
<td>the fixed timeout value for blocked transactions</td>
<td>50</td>
</tr>
<tr>
<td>msg-que-size</td>
<td>fixed size of message queues of the LAN network</td>
<td>5 – 100</td>
</tr>
<tr>
<td>loss</td>
<td>probability of message loss</td>
<td>0.000001</td>
</tr>
<tr>
<td>conf</td>
<td>probability of conflict due to concurrency</td>
<td>0.001</td>
</tr>
</tbody>
</table>

An event in the simulator is a quadruple: (event-type, event-time, trans-id, site). There are 5 independent event types in the simulator. These are, arrival, endcpu, endio, endnet, and timeout. Since the number of timeout events is not fixed, we could not use a purely static event list.

Each site is an instance of a large structure that includes the actual queues (cpu, io, net, and wait), the servers and their status, and all statistics needed. It also contains two counts for the input and output network queues in order to implement finiteness of message queues. One hack we used was to incorporate the input network queue with the cpu queue, yet count only messages that are put in the cpu queue from other sites.

The workload consists of jobs of different types. These are: transactions, subtransactions (read, write, or commit), and communication messages. The way we implemented jobs mimics a balloon that hops through queues and servers. Each time the job finishes a service, it shrinks, and uses its state information to know to where to go next. The criterion that the simulator used for termination is the number of finished transactions (committed + aborted).

Table 2 lists the simulator parameters and the corresponding values that are used in the experiment presented in the next subsection.
4.3.2 The Simulation Experiment

Statement of the Problem Finite network queues limit the maximum transaction load that can be applied to a distributed database system that uses expensive replication methods like the quorum consensus. This experiment studies the effect of network queue length on the maximum possible transaction load.

Procedure For each value of the network queue length, we monotonically decreased the mean inter-arrival time (increased the arrival rate) till the point where the simulator started to report dropped messages. Such a process took at least 10 repeated runs to reliably obtain a single point. We have repeated the experiment for two different values of the mean service time of the network. We actually doubled the speed of the network to test the scalability of the maximum transaction load (arrival rate) to the network mean service time.

Data We used the simulator with the parameter settings shown in Table 2. The network queue length was varied from 5 to 100. Figure 13 shows the maximum possible transaction load against a spectrum of maximum network queue lengths, for two mean network service times.

Discussion Figure 13 shows how the maximum arrival rate increases as the maximum message queue length increases from 5 to 100 entries. For a network mean service time of 0.50, doubling the maximum message queue length from 30 to 60 increases the maximum arrival rate by only 45%. To double, the maximum arrival rate, a maximum message queue length of 90 should be used. Response time in this case was unacceptable. For network mean
service time of 0.25. Doubling the maximum message queue length from 30 to 60 increases the maximum arrival rate by 64%. We observe that increasing the maximum message queue length does help increasing the maximum arrival rate. However, response time can highly increase for large queue lengths. The response time effect can be reduced by using faster networks.

5 Conclusion

We introduced algorithmic and operational availability as two complementary measures of reliability that should be examined simultaneously in order to successfully analyze fault-tolerance of distributed database systems.

We presented a customer-stationary availability model that includes transaction, data, configuration and failure parameters, and parameters pertinent to the replication control method itself. We used the model to study algorithmic availability of the read-one-write-all and a special case of the quorum consensus replication control methods. We used the availability model along with experimental performance data in studying the tradeoff between availability and performance when the degree of replication is varied. We defined and determined the practical degree of replication which is the smallest degree of replication at which certain performance and availability requirements are realized.

We studied the availability and performance of a 9-site RAID system. We found that for 0.90 site reliability, the read-one-write-all replication control method has practical degree of replication of 2 copies, for update percents less than 50%. We have shown that higher degrees of replication are not useful for the case of read-only transactions and are restrictive for the 20% update percent case. For higher update percents, we found that replication is not useful at all.

Under the same assumptions and for similar reliability assumptions, we studied the quorum consensus method with read and write majority. We call this method quorum consensus where reads are same as writes, or QC-RSW. We found that for 50% update percent, the practical degree of replication was 5 copies. For higher site reliability (0.95, instead of 0.90), the practical degree of replication was improved to 3 copies. As predicted by the model, availability did not change when the update percent was varied since QC-RSW requires the same number of sites for both the read and write operations. We observed that the availability of the QC-RSW has an interesting behavior. The availability oscillated around a slowly increasing average that reached a plateau at high degrees of replication (greater than 7 copies). We found that Even degrees of replication consistently had less availability than the preceding odd degrees of replication. This excludes Even degrees of replication to ever be best choices as practical degrees of replication for the QC-RSW method. This behavior is explained in Section 3.3.4.

To complement our study of the read-one-write-all and the quorum consensus meth-
ods, we studied the operational availability of these methods using the SETH distributed
database system prototype and using an event-driven simulation. We performed an exper­
iment to measure the maximum transaction load for which the prototype system SETH
can be operable under the same quorum consensus replication control method. The study
was performed on the Sequent Symmetry machine which can process about 512-bytes long
messages at 200 message/second rate. We found that, for arrival rates greater than 2.5,
the message traffic approaches the maximum message processing rate, at which point UDP
sockets start dropping messages, due to the finiteness of its input message queues. We also
observed that the traffic does not only remain saturated after this point but, instead slightly
drops off. The drop off is because some of the dropped messages are actually request mes­
sages for which some transactions will be waiting on their replies. Since replies will never
arrive, some transactions will be blocked and will never contribute to further message traffic.

As an extension to the SETH experiment, we used the event-driven simulation to in­
vestigate the scalability of transaction processing power to increasing the maximum input
message queue length of the UDP sockets. We observed that the maximum arrival rate does
not scale linearly with the maximum input message queue length. Moreover, we found that
increasing input message queue size resulted in an unacceptable response time. We finally
showed that resorting to larger communication bandwidth scales well and in the same time
improves response time.

Acknowledgements

The authors wish to thank other members of the RAID group: John Riedl, Srinivasan
Jagannathan, and Chuck Koelbel for their contribution to the RAID experimental facilities
and the development of the SETH software. The authors are also grateful to Professor
Wojciech Szpankowski for his useful comments on an earlier draft of section 2. Finally, the
authors would like to thank the anonymous referees for their valuable suggestions that helped
us revising the availability model in section 2.4 and that improved the presentation of the
paper.

References

Proceedings of the 2nd Int'l Conference on Software Engineering, pages 562–570,


31